Chapter 3:
Geometric Transformations
-Rotation-

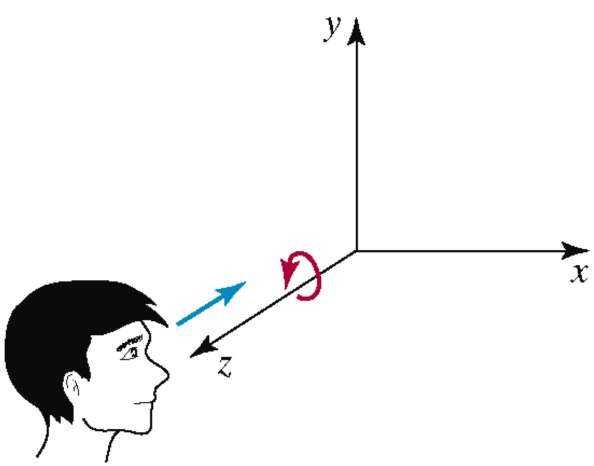
Sang Il Park

Dept. of Software

Examples of Affine Transformations

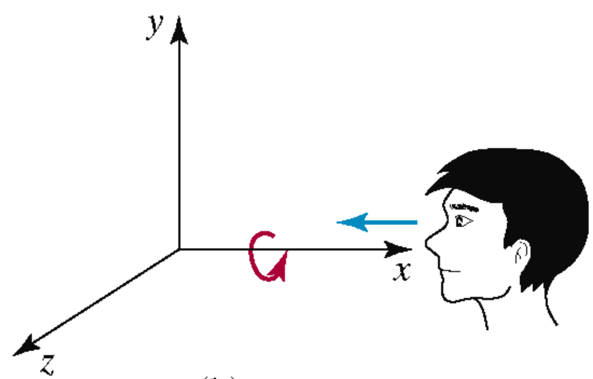
- 3D rotation

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



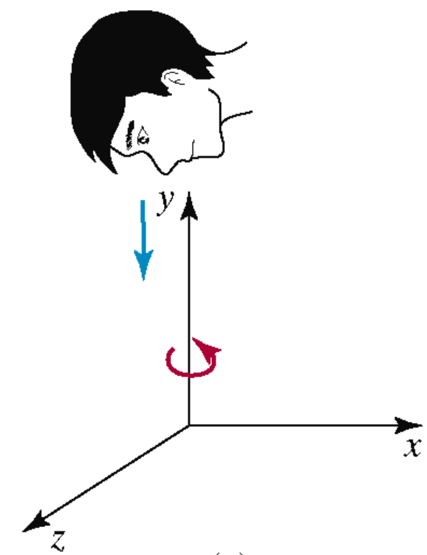
(a)

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



(b)

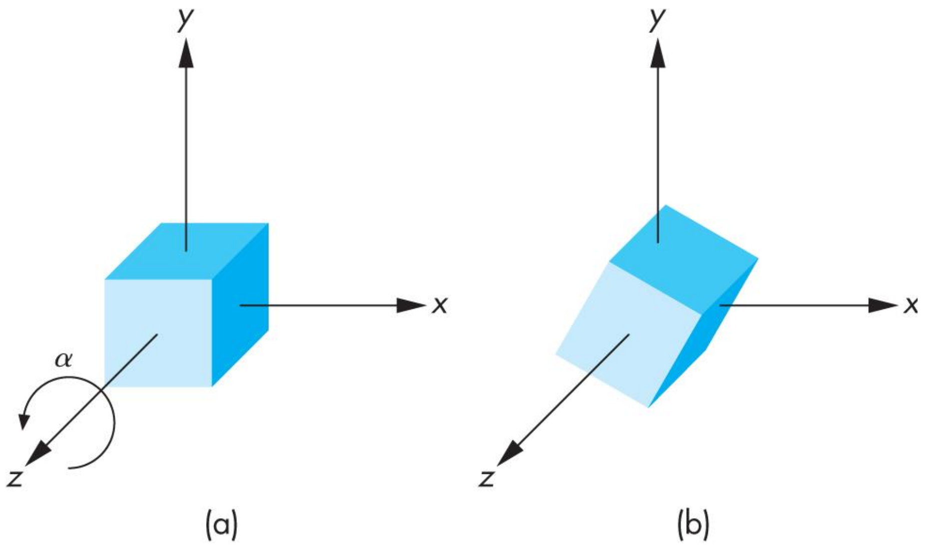
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



(c)

3D Rotation Matrix about Z Axis

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



3D Rotation about $\mathbf{x}$ and $\mathbf{y}$ axes

- Same argument as for rotation about *z* axis
  - For rotation about x axis, x is unchanged
  - For rotation about y axis, y is unchanged

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

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

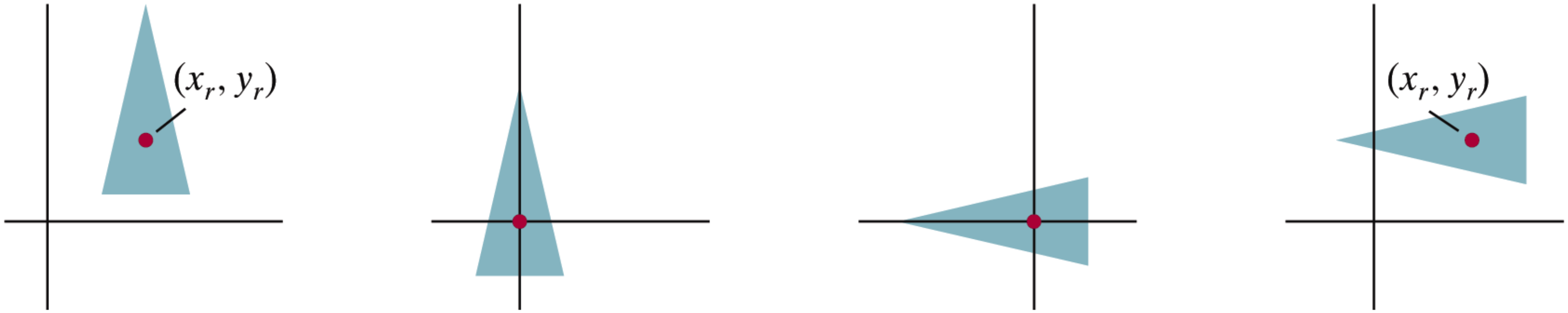
2D Pivot-Point Rotation

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

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

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

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



(a) (b) (c) (d)

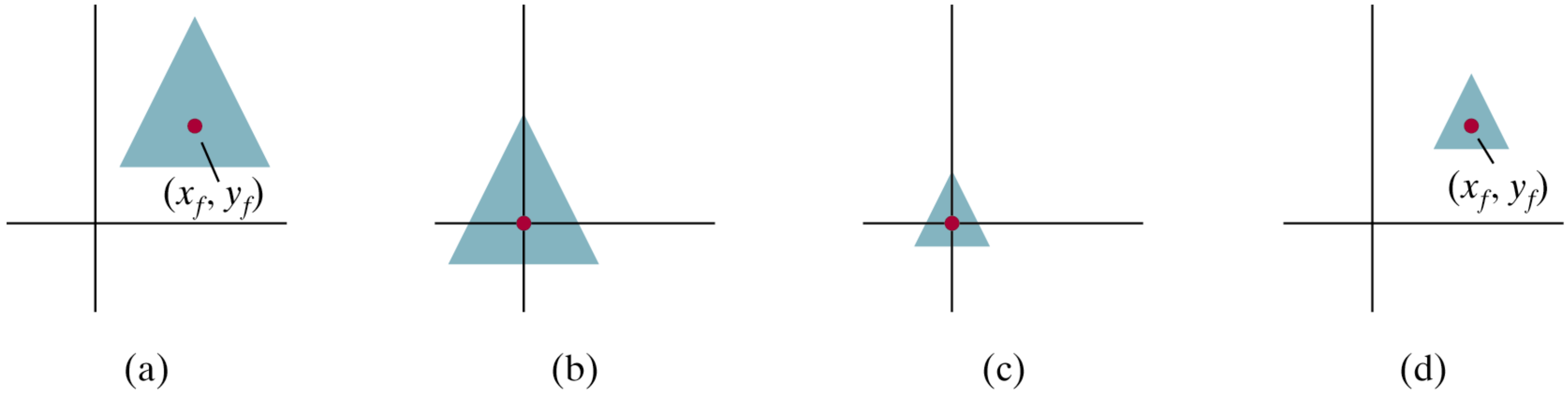
2D Fixed-Point Scaling

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

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

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

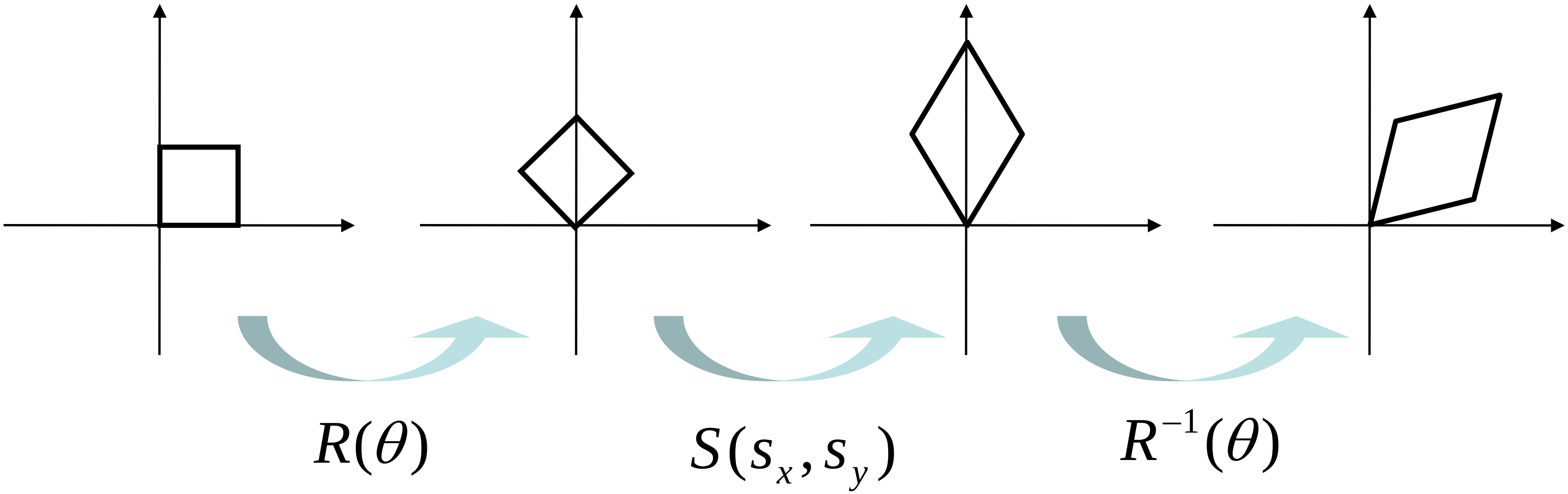
$$=\begin{pmatrix}s_x & 0 & (1-s_x)\cdot x\\ 0 & s_y & (1-s_y)\cdot y\\ 0 & 0 & 1\end{pmatrix}$$



Scaling Direction

- Scaling along an arbitrary axis

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



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

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

$$RR^T = R^T R = I$$ (Distance preserving)

$$\det R = 1$$ (No reflection)

Rigid Body Rotation

- R is normalized
  - The squares of the elements in any row or column sum to 1

$$R R^T = R^T R = I$$

- R is orthogonal
  - 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

- How to rotate around **u** vector (**u** = given rotation axis)

➔ Rotate about x and y axes to make **u** align with the *z*-axis

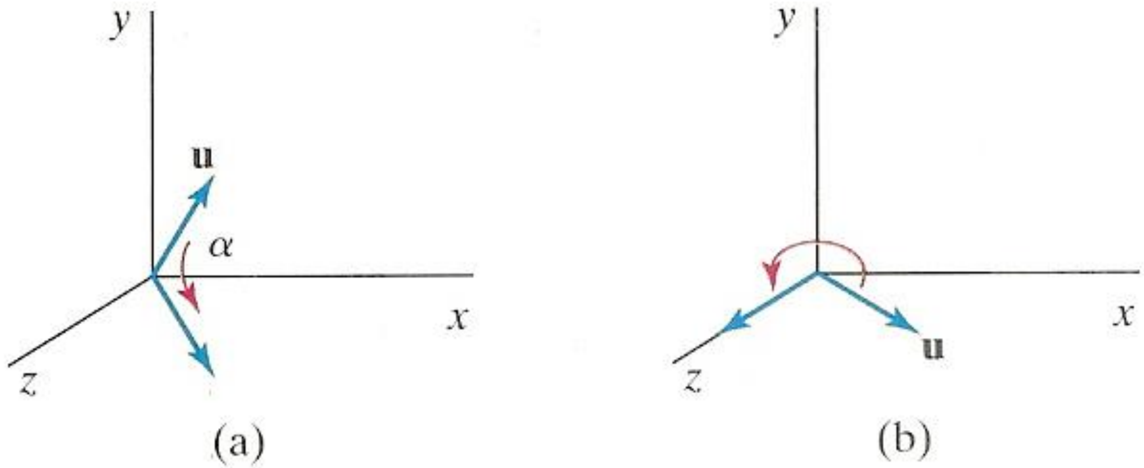


FIGURE 5-45 Unit vector **u** is rotated about the x axis to bring it into the xz plane (a), then it is rotated around the y axis to align it with the z axis (b).

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-asis

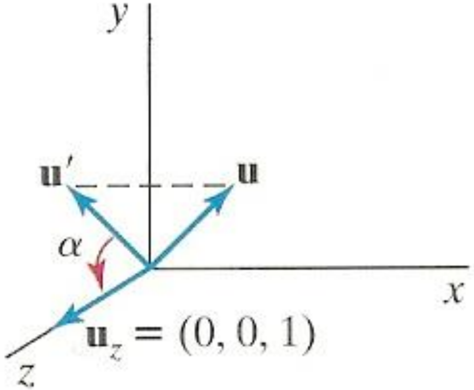


FIGURE 5-46 Rotation of **u** around the x axis into the xz plane is accomplished by rotating $\mathbf{u}'$ (the projection of **u** in the yz plane) through angle α onto the z axis.

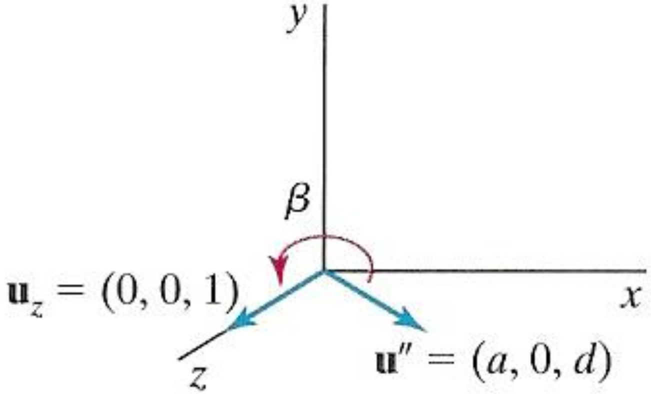


FIGURE 5-47 Rotation of unit vector $\mathbf{u}''$ (vector **u** after rotation into the xz plane) about the y axis. Positive rotation angle β aligns $\mathbf{u}''$ with vector $\mathbf{u}_z$.

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)

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

$$\begin{aligned} \mathbf{u}' \times \mathbf{u}_z &= \mathbf{u}_x \|\mathbf{u}'\|\,\|\mathbf{u}_z\| \sin\alpha \\ &= \mathbf{u}_x \cdot b \end{aligned} \quad \Rightarrow \quad \sin\alpha = \frac{b}{\|\mathbf{u}'\|\,\|\mathbf{u}_z\|} = \frac{b}{\sqrt{b^2 + c^2}}$$

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

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

  - 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]

Gimble

- Hardware implementation of Euler angles
- Aircraft, Camera

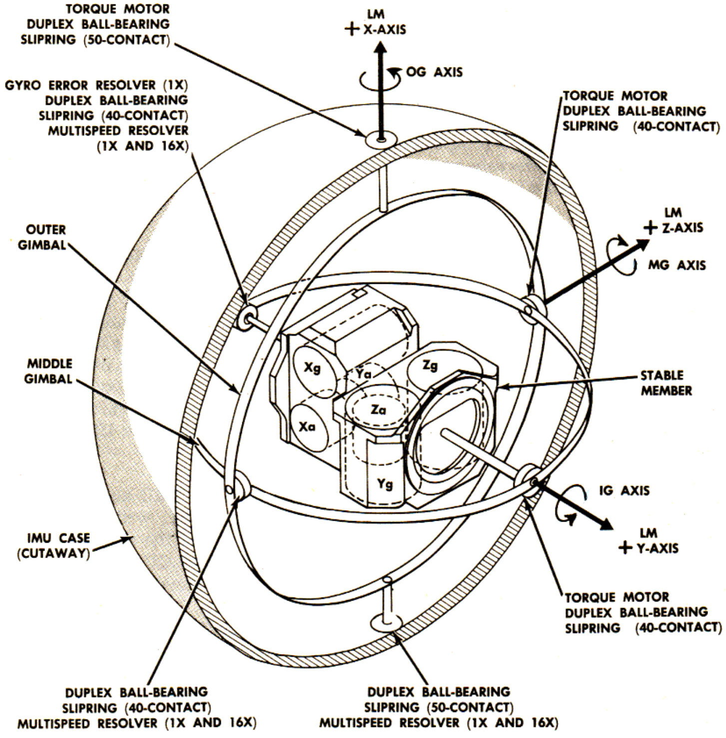
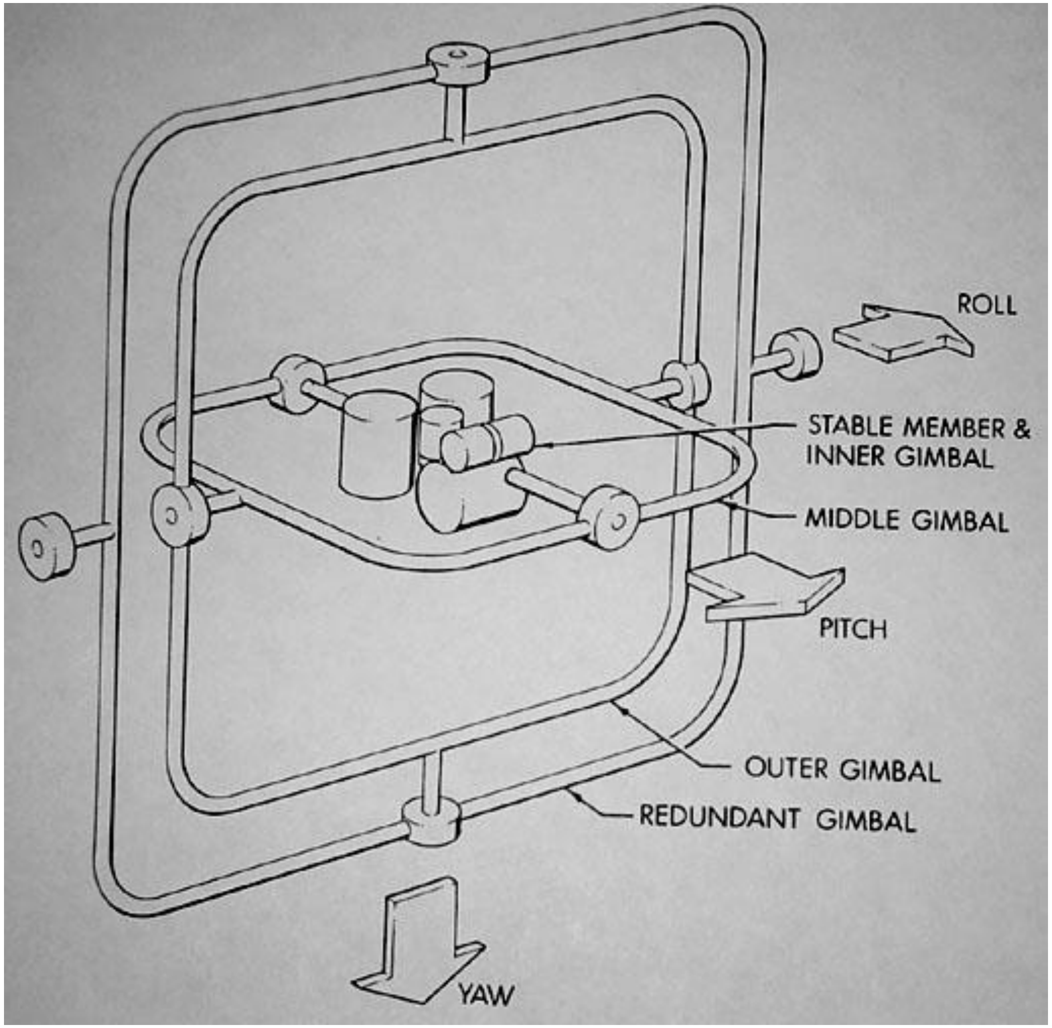


Figure 2.1-24. IMU Gimbal Assembly

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

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

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

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

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.

Smooth Rotation

- Create transformations from $\mathbf{M_0}$ to $\mathbf{M_n}$ ***smoothly***
  - Problem: find a sequence of model-view matrices $\mathbf{M_0,M_1,\ldots\ldots,M_n}$ for each frame
    to see a smooth transition

- One solution for rotation (using Euler angles):
  - Find $\mathbf{R_0 = R_{0z}\,R_{0y}\,R_{0x}}$ and $\mathbf{R_n = R_{nz}\,R_{ny}\,R_{nx}}$
  - Then, Create a sequence of rotation $\mathbf{R_0,R_1,\ldots\ldots,R_n}$:
    $\mathbf{R_i = R_{iz}\,R_{iy}\,R_{ix}}$ (where, ix, iy, iz is the interpolated angles from the beginning and the end)
  - Not very effective!
  - Quaternions can do it better!

Quaternions

- Extension of imaginary numbers from two to three dimensions
- Requires one real and three imaginary components $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$

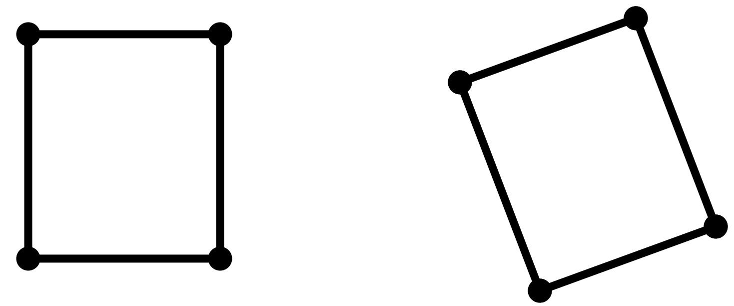
$$q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$$

- 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

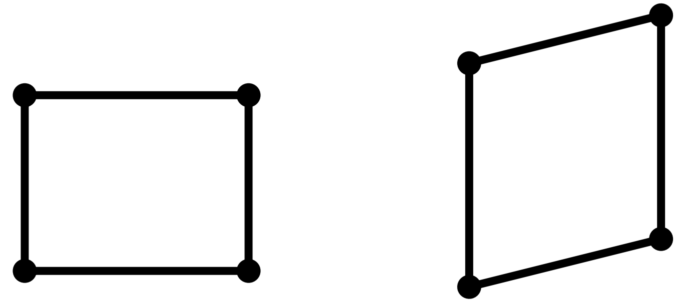
Computer Animation 수업에서 다룹니다.

Taxonomy of Transformations

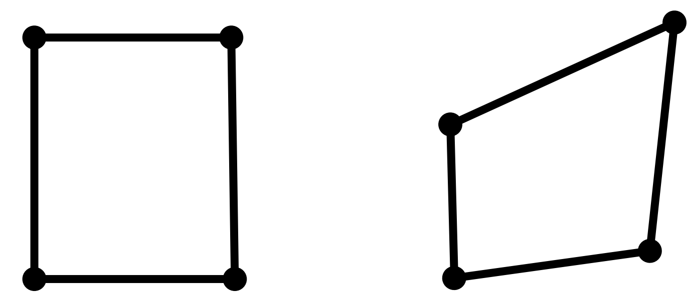
Rigid



Affine



Projective



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

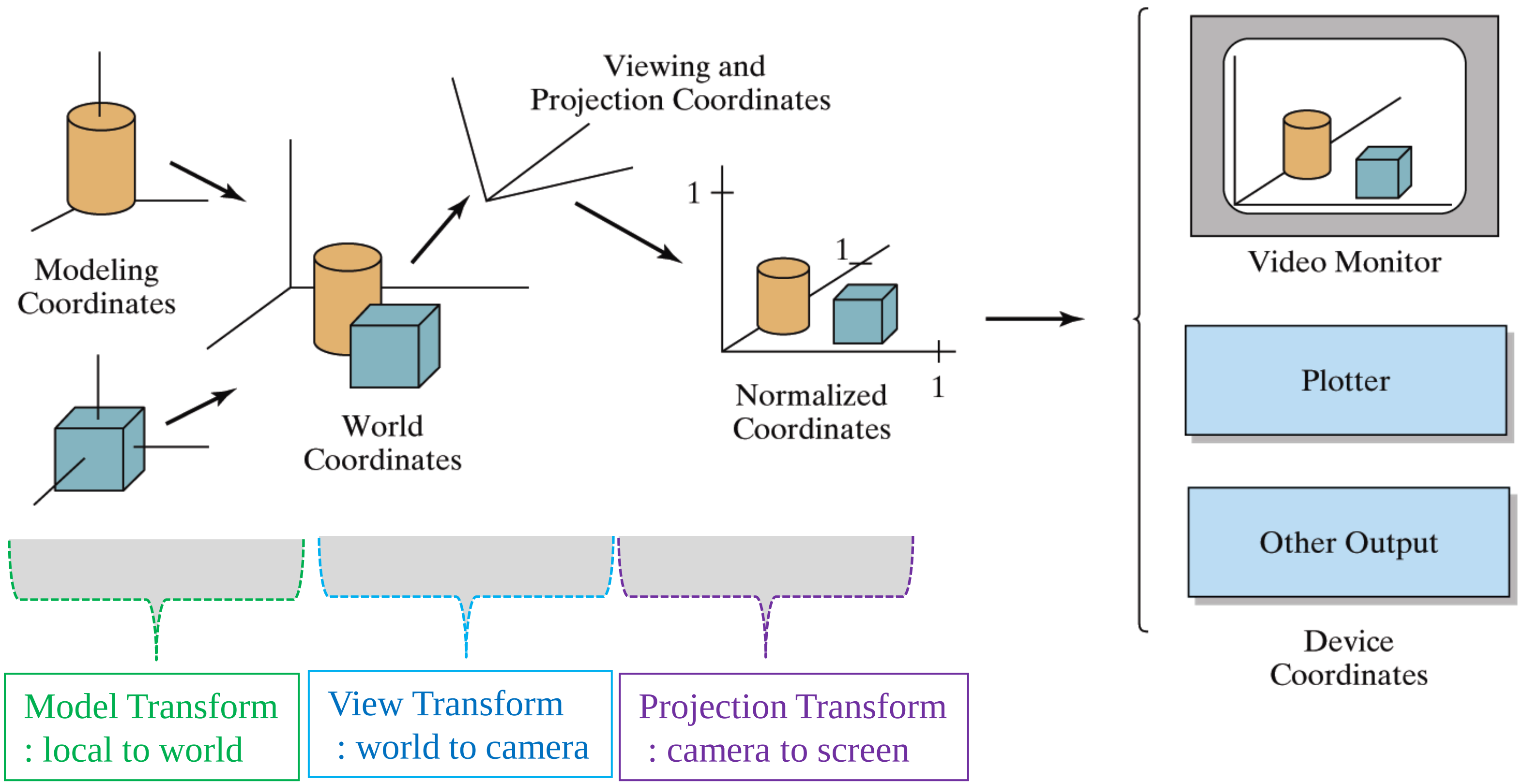
$$
\begin{aligned}
T &= \mathbf{R}(\theta_2)\cdot\mathbf{R}(\theta_1) \\
&= \mathbf{R}(\theta_2+\theta_1)
\end{aligned}
$$

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

OpenGL Geometric Transformations

OpenGL Geometric Transformations

- **Consecutive Transformations in OpenGL Pipeline**



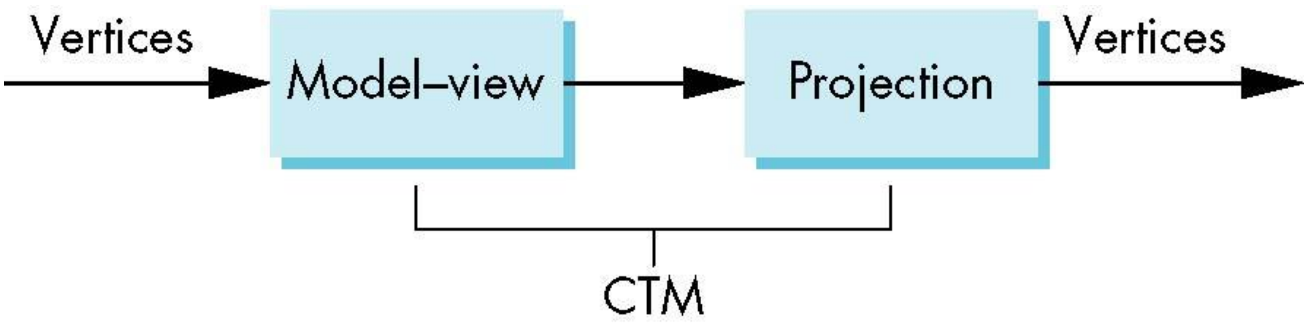
OLD OpenGL Matrices

- Two types of predefined transformations (matrices)
  - Model-View (**`GL_MODELVIEW`**) : model+view
  - Projection (**`GL_PROJECTION`**)

- Single set of functions for manipulation
- Select which to manipulated by
  - **`glMatrixMode(GL_MODELVIEW);`**
  - **`glMatrixMode(GL_PROJECTION);`**

Current Transform Matrix (CTM) in OLD OpenGL

- OpenGL had a model-view and a projection matrix in the pipeline which were concatenated together to form the CTM
- We will emulate this process



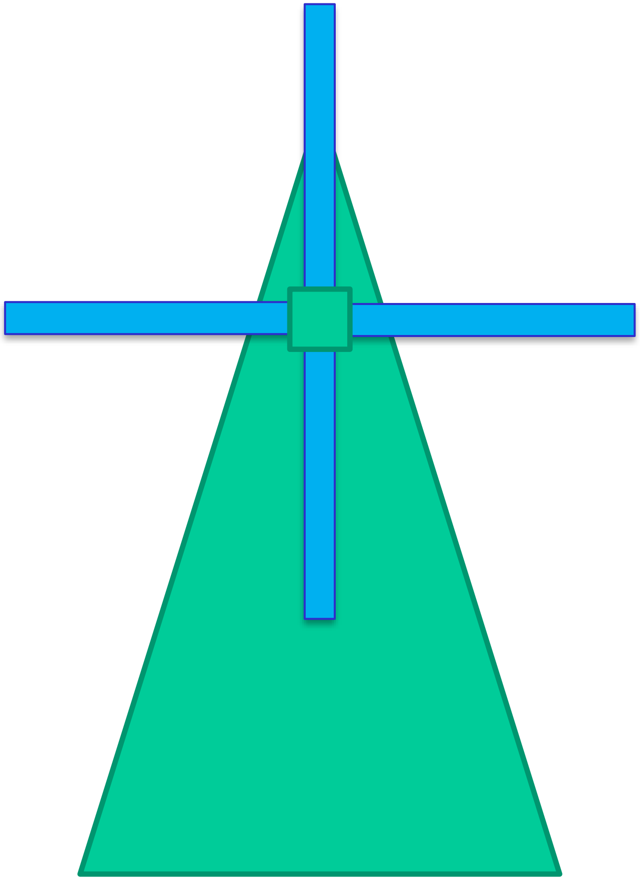
CTM: Current Transform Matrix

OLD OpenGL Geometric Transformations

- Basic Transpormation:
  - **`glLoadIdentity();`**
  - **`glTranslatef(tx, ty, tz);`**
  - **`glRotatef(theta, vx, vy, vz);`** **angle-axis**
    - **`(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
  - **Column major**

Model Transformation

연습: 바람개비(풍차)만들기



Instance Transformation

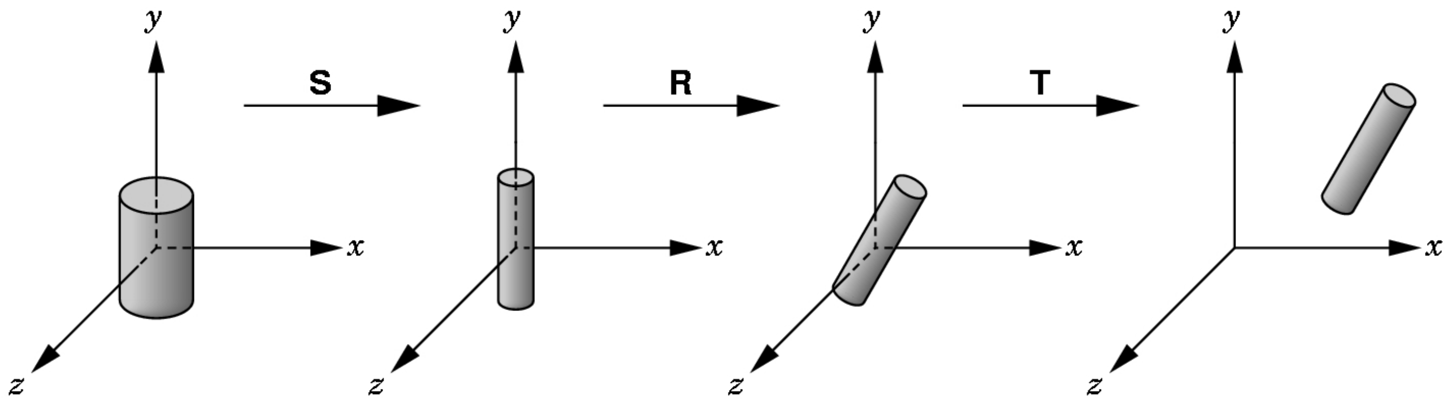
- Often we need several instances of an object
  - Wheels of a car
  - Arms or legs of a figure
  - Chess pieces

Instance Transformation

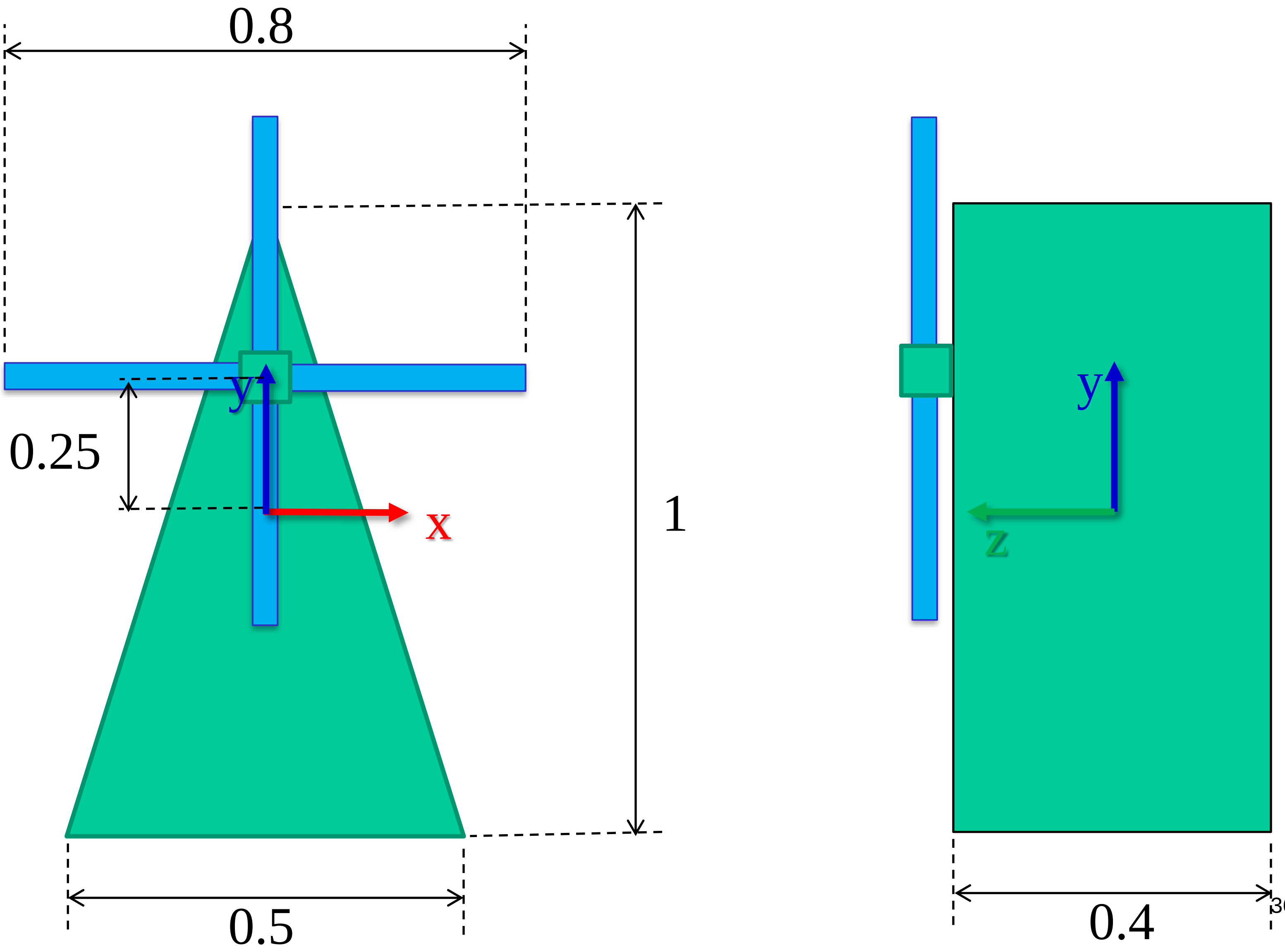
- Instances can be shared across space or time
- Write a function that renders the object in "standard" configuration
- Apply transformations to different instances
- Typical order: *scaling → rotation → translation*

Sample Instance Transformation (old style)

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(...);
glRotatef(...);
glScalef(...);
gluCylinder(...);
```



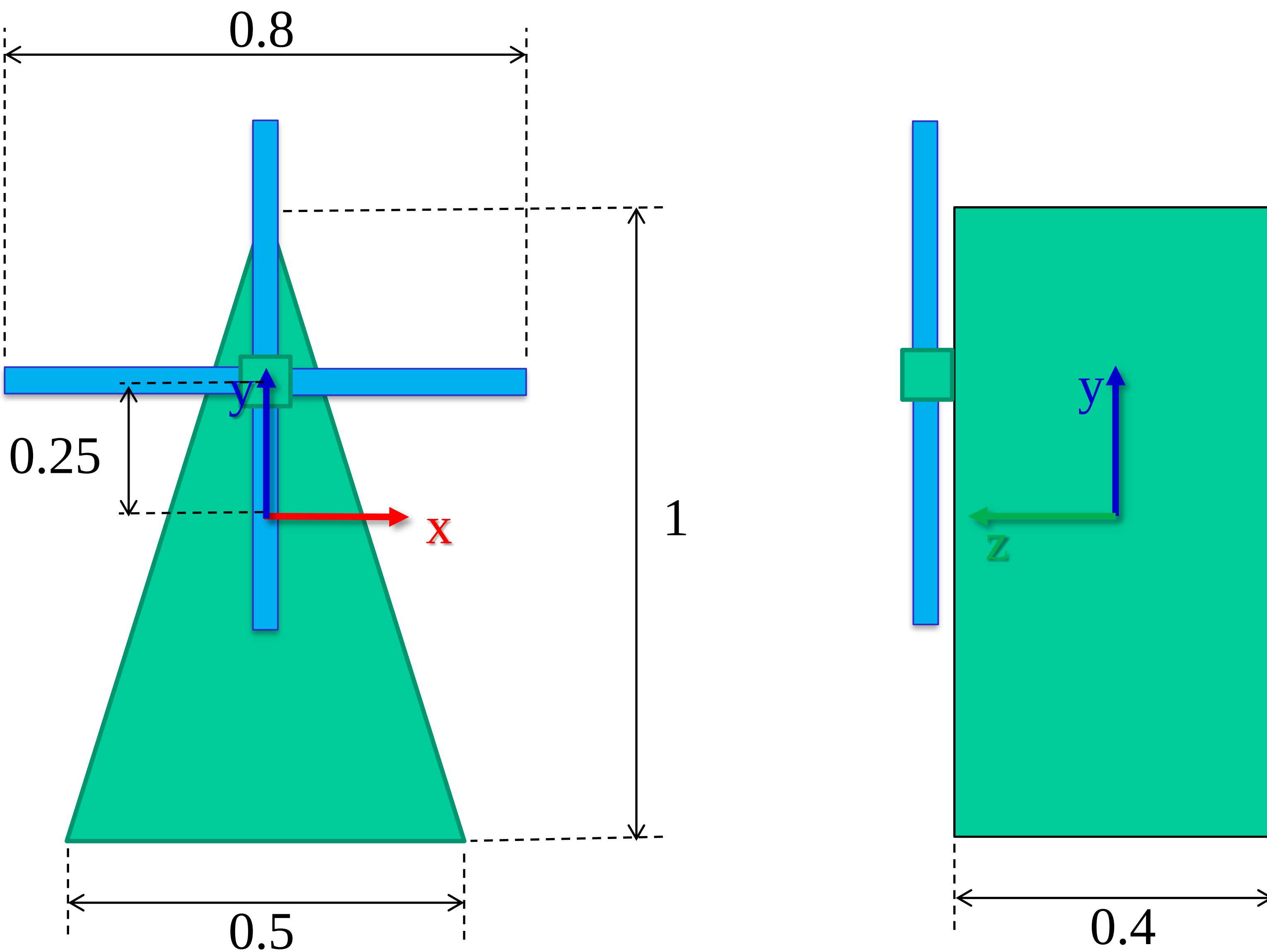
구체적인 계획



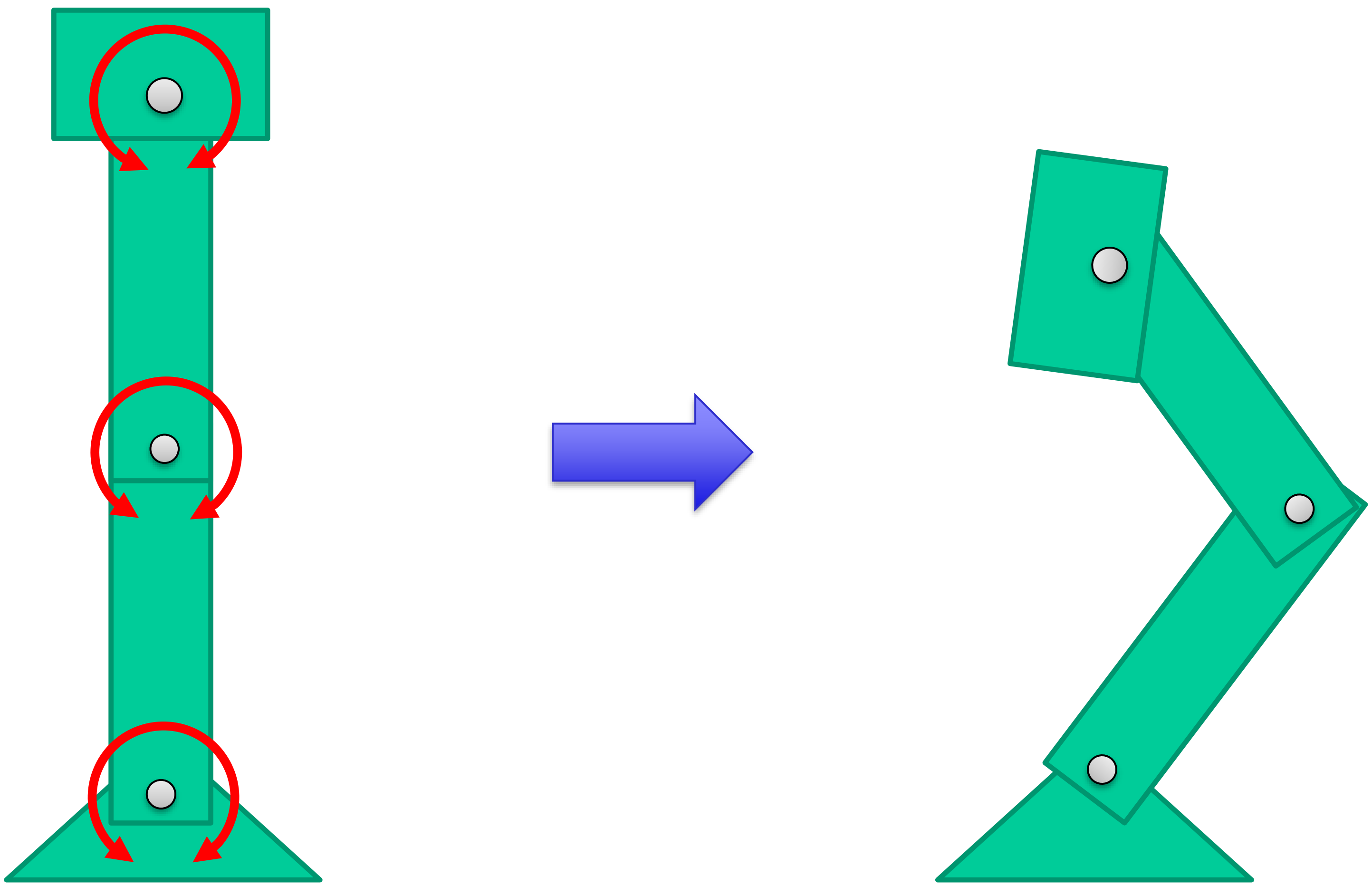
Instance Transformation

- Instances can be shared across space or time
- Write a function that renders the object in "standard" configuration
- Apply transformations to different instances
- Typical order: *scaling* → *rotation* → *translation*

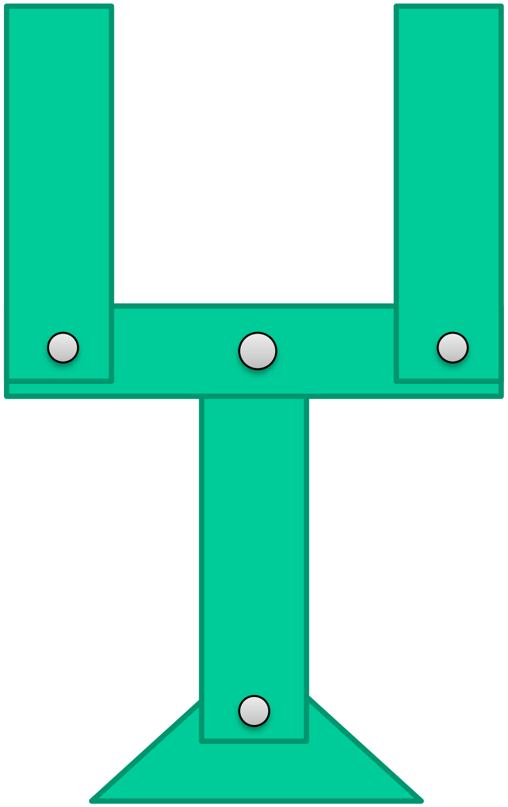
구체적인 계획



로보트 팔 만들기

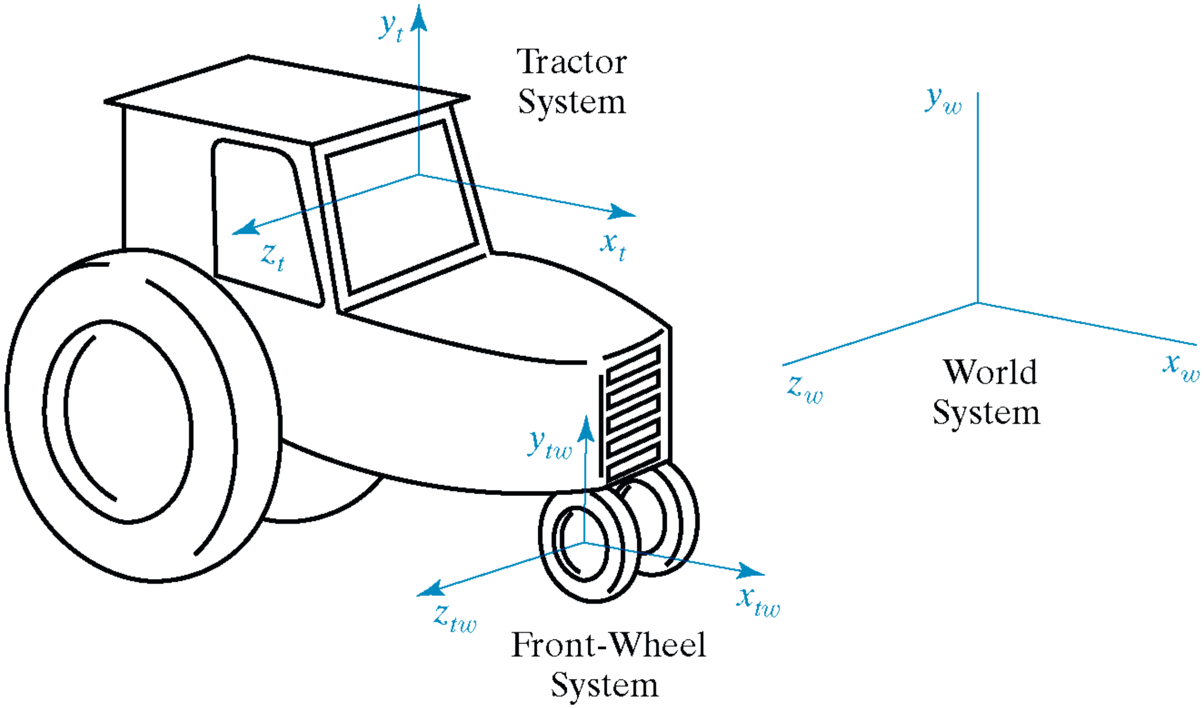


로보트 팔 만들기2



Hierarchical Modeling

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



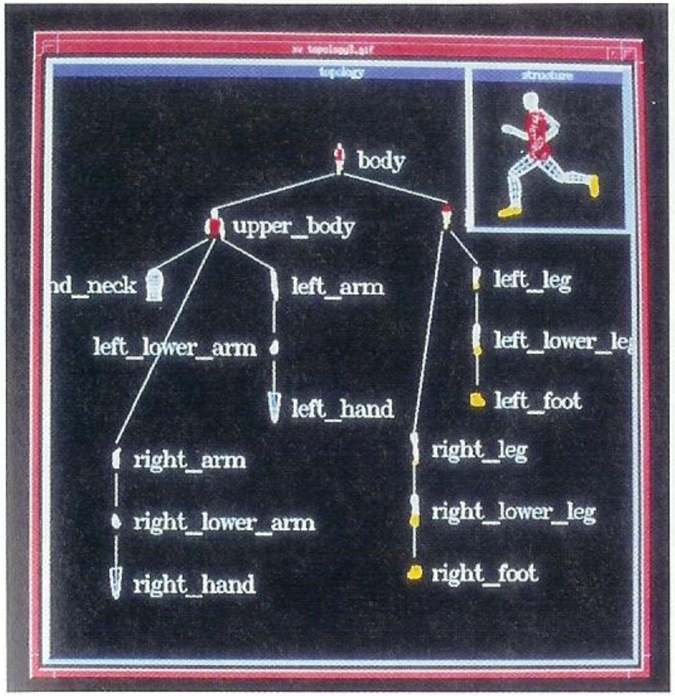


FIGURE 14-4 An object hierarchy generated using the PHIGS Toolkit package developed at the University of Manchester. The displayed object tree is itself a PHIGS structure. (*Courtesy of T. L. J. Howard, J. G. Williams, and W. T. Hewitt, Department of Computer Science, University of Manchester, United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

OpenGL Matrix Stacks (OLD)

- 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

Matrix Stacks by your own

- We emulate Matrix Stacks by using:
  - Linked List such as ***std::list*** or ***std::deque***
  - Or a tree structure for more generality.