Geometric Transformations

Ceng 477 Introduction to Computer Graphics
Fall 2007
Computer Engineering
METU

2D Geometric Transformations

Basic Geometric Transformations

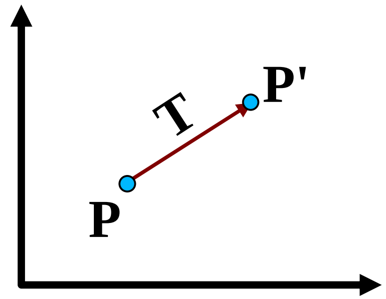
- Geometric transformations are used to transform the objects and the camera in a scene (for animation or modelling) and are also used to transform World Coordinates to View Coordinates

- Given the shape, transform all the points of the shape? Transform the points and/or vectors describing it.
- For example:
  Polygon: corner points
  Circle, Ellipse: center point(s), point at angle 0
- Some transformations preserves some of the attributes like sizes, angles, ratios of the shape.

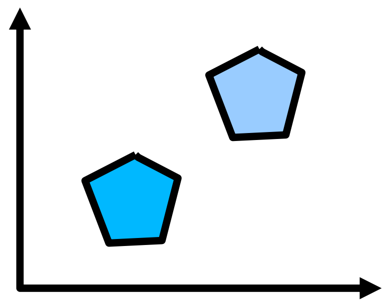
Translation

- Simply move the object to a relative position.

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



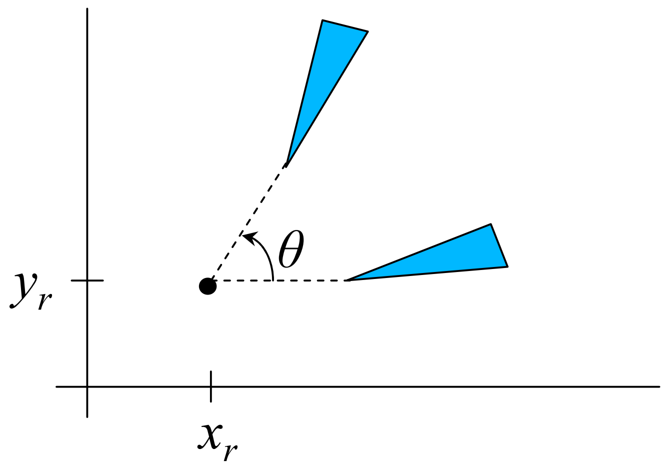
$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad \mathbf{P}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$$



$$\mathbf{P}' = \mathbf{P} + \mathbf{T}$$

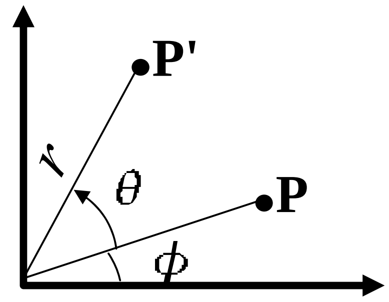
Rotation

- A rotation is defined by a rotation axis and a rotation angle.
- For 2D rotation, the parameters are rotation angle (θ) and the rotation point (x_r, y_r).
- We reposition the object in a circular path arround the rotation point (pivot point)



Rotation

- When $(x_r, y_r) = (0,0)$ we have



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

The original coordinates are:

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

Substituting them in the first equation we get:

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

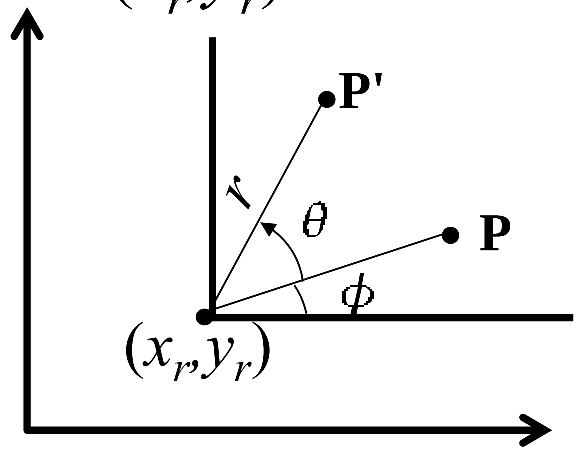
In the matrix form we have: $\mathbf{P}' = \mathbf{R}\cdot\mathbf{P}$

where $\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

Rotation

- Rotation around an arbitrary point (x_r, y_r)

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$
$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$$

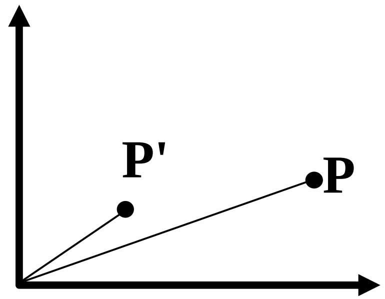


- This equations can be written as matrix operations (we will see when we discuss homogeneous coordinates).

Scaling

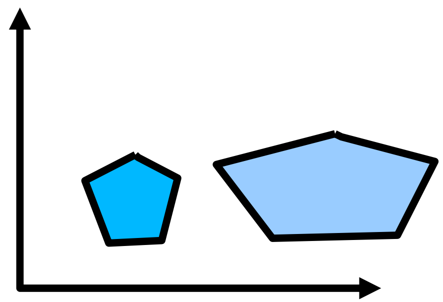
- Change the size of an object. Input: scaling factors (s_x, s_y)

$$x' = xs_x \qquad y' = ys_y$$



$$\mathbf{S} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$



non-uniform vs.
uniform scaling

Homogenous Coordinates

- All transformations can be represented by matrix operations.
- Translation is additive, rotation and scaling is multiplicative (+ additive if you rotate around an arbitrary point or scale around a fixed point); making the operations complicated.
- Adding another dimension to transformations make translation also representable by multiplication. Cartesian coordinates vs homogenous coordinates.

$$x = \frac{x_h}{h} \quad y = \frac{y_h}{h} \quad P = \begin{bmatrix} x_h \\ y_h \\ h \end{bmatrix} = \begin{bmatrix} h \cdot x \\ h \cdot y \\ h \end{bmatrix}$$

- Many points in homogenous coordinates can represent the same point in Cartesian coordinates.
- In homogenous coordinates, all transformations can be written as matrix multiplications.

Transformations in Homogenous C.

- Translation

$$T(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = T(t_x, t_y) \cdot P$$

- Rotation

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

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

- Scaling

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

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

Composite Transformations

- Application of a sequence of transformations to a point:

$$\mathbf{P}' = \mathbf{M}_2 \cdot \mathbf{M}_1 \cdot \mathbf{P}$$
$$= \mathbf{M} \cdot \mathbf{P}$$

Composite Transformations

- First: composition of similar type transformations
- If we apply to successive translations to a point:

$$\mathbf{P}' = \mathbf{T}(t_{2x}, t_{2y}) \cdot \{\mathbf{T}(t_{1x}, t_{1y}) \cdot \mathbf{P}\}$$

$$= \{\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y})\} \cdot \mathbf{P}$$

$$T(t_{2x}, t_{2y}) \cdot T(t_{1x}, t_{1y}) = \begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix} = T(t_{1x} + t_{2x}, t_{1y} + t_{2y})$$

Composite Transformations

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

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

$$\mathbf{S}(s_{2x},s_{2y})\cdot\mathbf{S}(s_{1x},s_{1y})=\begin{bmatrix}s_{2x} & 0 & 0\\ 0 & s_{2y} & 0\\ 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}s_{1x} & 0 & 0\\ 0 & s_{1y} & 0\\ 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}s_{1x}\cdot s_{2x} & 0 & 0\\ 0 & s_{1y}\cdot s_{2y} & 0\\ 0 & 0 & 1\end{bmatrix}=\mathbf{S}(s_{1x}\cdot s_{2x},s_{1y}\cdot s_{2y})$$

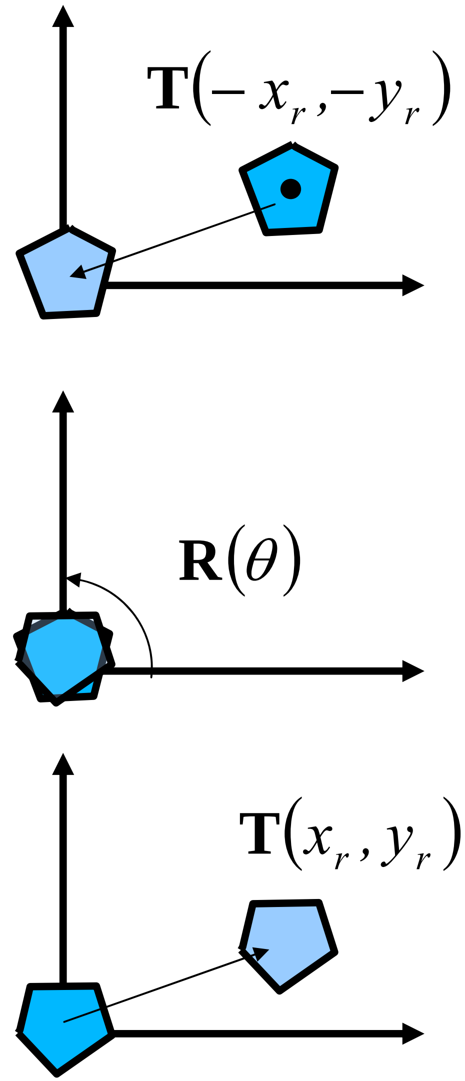
Rotation around a pivot point

- Translate the object so that the pivot point moves to the origin
- Rotate around origin
- Translate the object so that the pivot point is back to its original position

$$\mathbf{T}(x_r, y_r)\cdot\mathbf{R}(\theta)\cdot\mathbf{T}(-x_r, -y_r) =$$

$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix}\cdot\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}\cdot\begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta)+y_r\sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta)-x_r\sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$



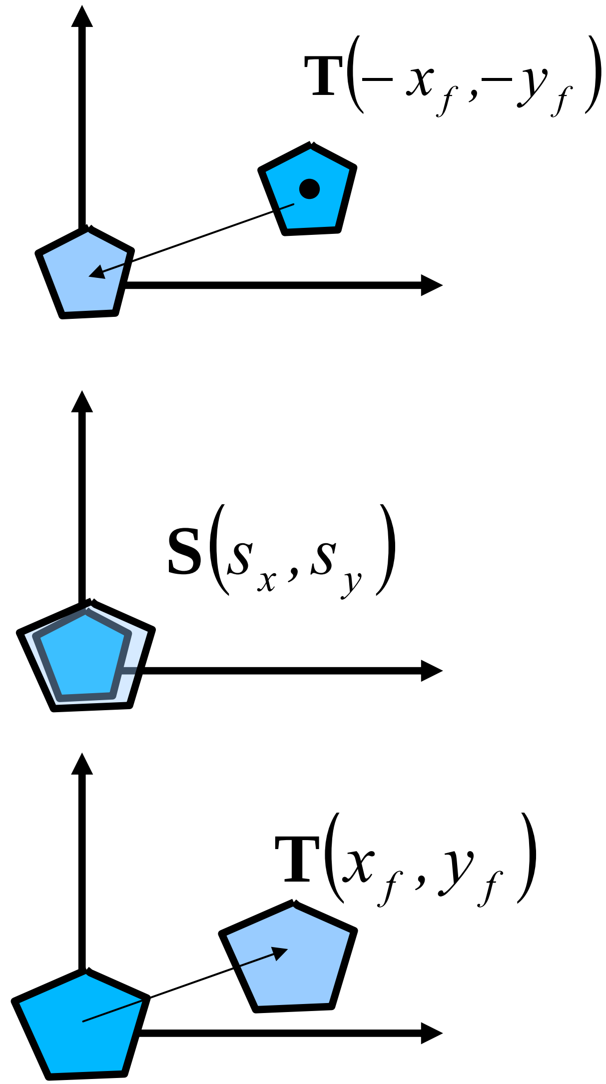
Scaling with respect to a fixed point

- Translate to origin
- Scale
- Translate back

$$\mathbf{T}(x_f, y_f) \cdot \mathbf{S}(s_x, s_y) \cdot \mathbf{T}(-x_f, -y_f) =$$

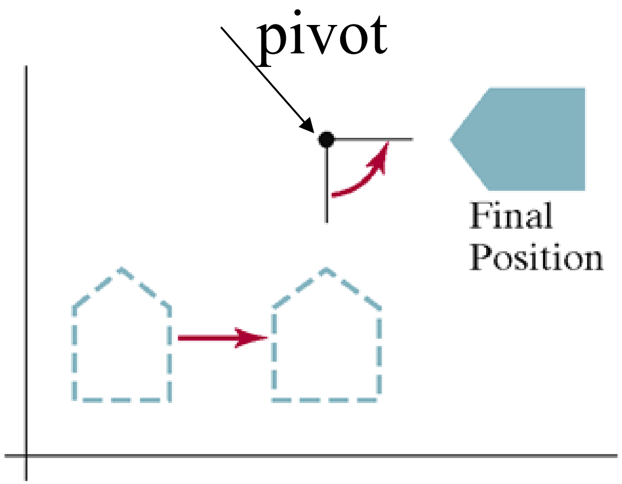
$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} =$$

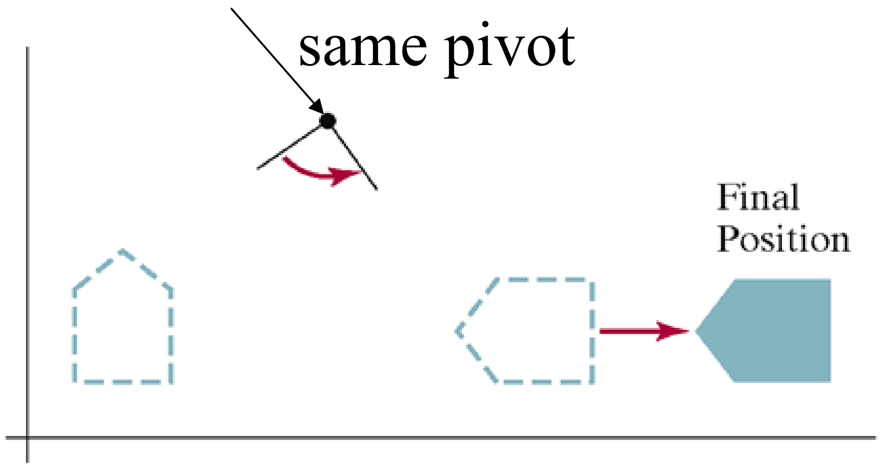
$$\begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix}$$



Order of matrix compositions

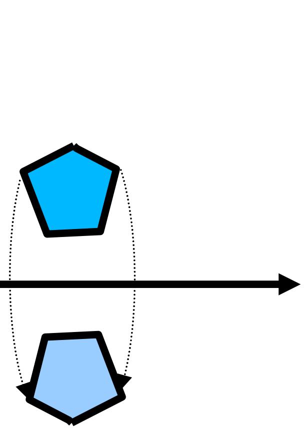
- Matrix composition is not commutative. So, be careful when applying a sequence of transformations.



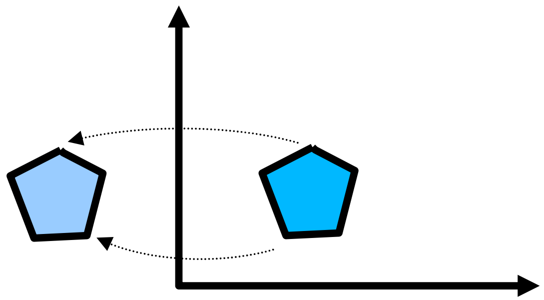


Other Transformations

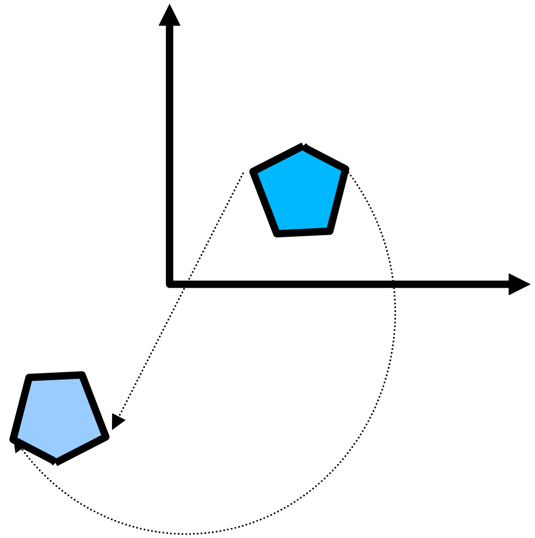
- Reflection



$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

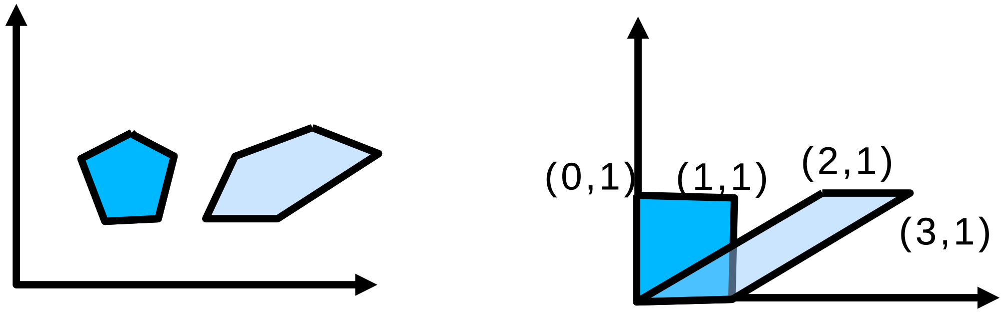


$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Shear: Deform the shape like shifted slices.

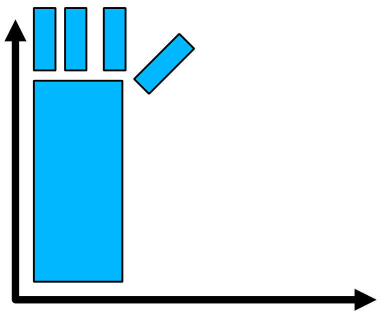


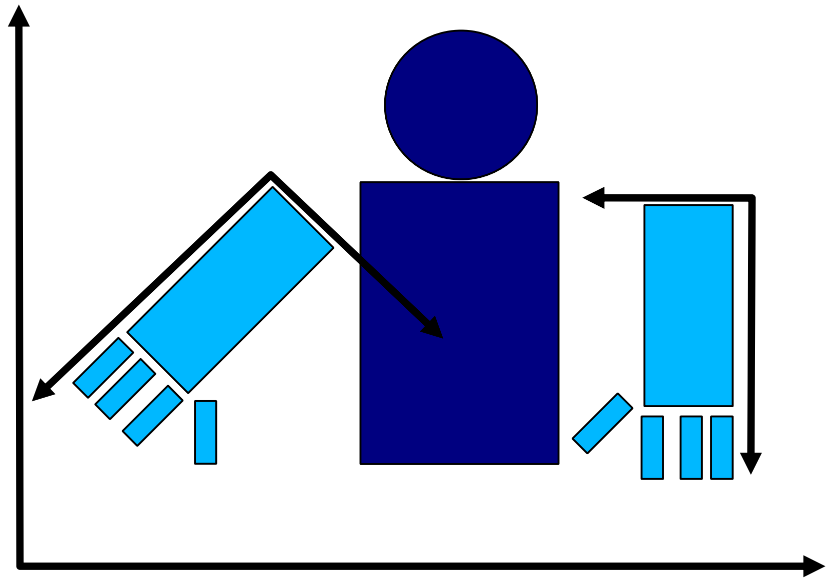
$$x' = x + sh_x \cdot y \qquad y' = y$$

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformations Between the Coordinate Systems

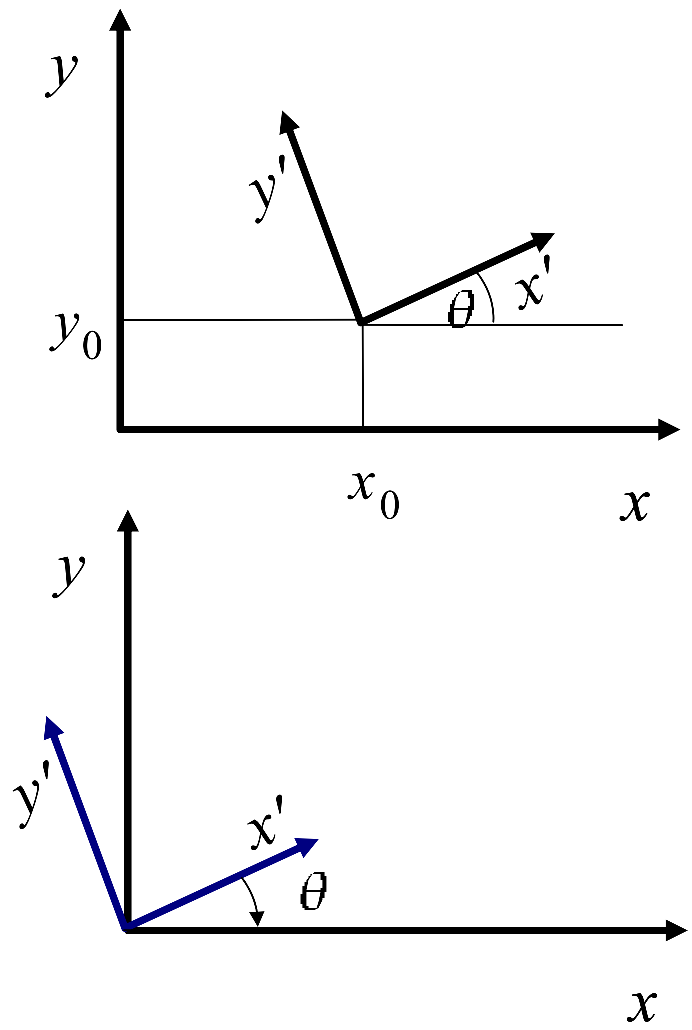
- Between different systems: Polar coordinates to cartesian coordinates
- Between two cartesian coordinate systems. For example, relative coordinates or window to viewport transformation.





- How to transform from *x,y* *to x',y' ?*
- *Superimpose x',y' to x,y*
- *Transformation:*
  - *Translate so that* (x_0, y_0) *moves to (0,0) of x,y*
  - *Rotate x' axis onto x axis*

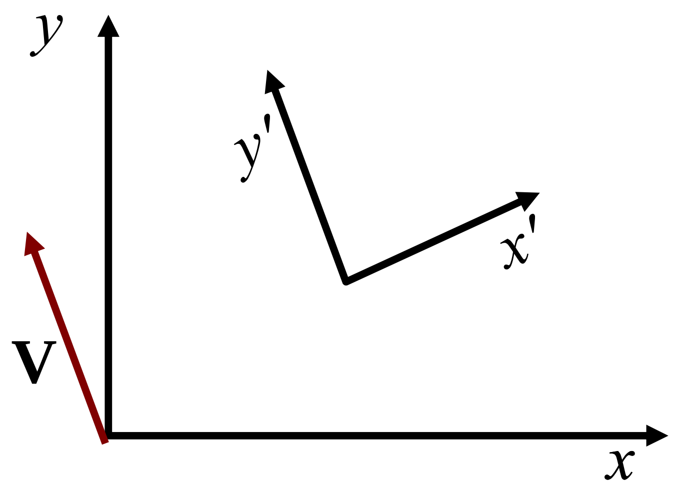
$$R(-\theta)\cdot T(-x_{0,}\,-y_0)$$



- Alternate method for rotation: Specify a vector $\mathbf{V}$ for positive y' axis:

unit vector in the y' direction :

$$\mathbf{v} = \frac{\mathbf{V}}{|\mathbf{V}|} = (v_x, v_y)$$



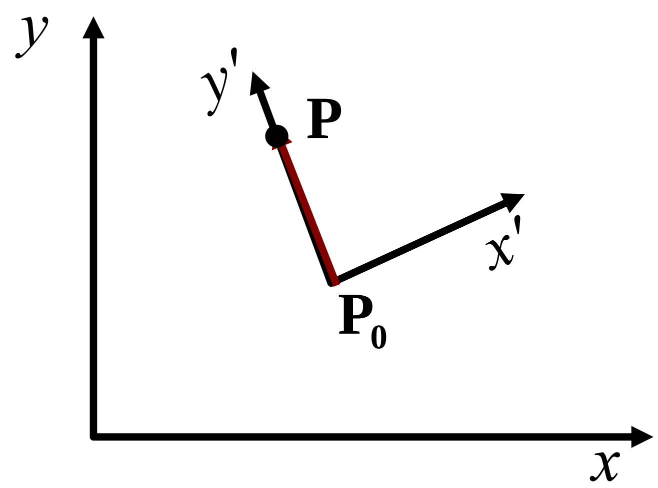
unit vector in the x' direction, rotate $\mathbf{v}$ clockwise 90°

$$\mathbf{u} = (v_y, -v_x) = (u_x, u_y)$$

- Elements of any rotation matrix can be expressed as elements of a set of orthogonal unit vectors:

$$\mathbf{R} = \begin{bmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} v_y & -v_x & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{v} = \frac{\mathbf{P} - \mathbf{P_0}}{\left|\mathbf{P} - \mathbf{P_0}\right|}$$

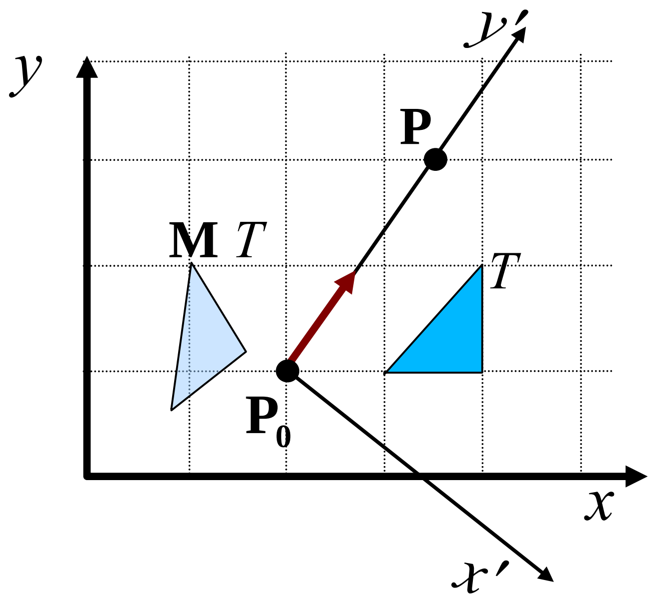


- Example:

$$\mathbf{P}_0 = (2,1) \quad \mathbf{P} = (3.5,3)$$

$$\mathbf{v} = \frac{\mathbf{P}-\mathbf{P_0}}{|\mathbf{P}-\mathbf{P}_0|} = \frac{(1.5,2)}{\sqrt{1.5^2+2^2}} = \left(\frac{1.5}{2.5}, \frac{2}{2.5}\right) = (0.6,0.8)$$

$$\mathbf{u} = (0.8,-0.6)$$



$$\mathbf{M} = \mathbf{R}(\mathbf{u},\mathbf{v})\cdot\mathbf{T}(-2,-1) =$$

$$\begin{bmatrix} 0.8 & -0.6 & 0 \\ 0.6 & 0.8 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.8 & -0.6 & -1 \\ 0.6 & 0.8 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

Let triangle T be defined as three column vectors: $\begin{bmatrix} 3 & 4 & 4 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

$$\mathbf{M}\cdot T = \begin{bmatrix} 0.8 & 1 & 1.6 \\ 0.6 & 2 & 1.2 \\ 1 & 1 & 1 \end{bmatrix}$$

Affine Transformations

- Coordinate transformations of the form:

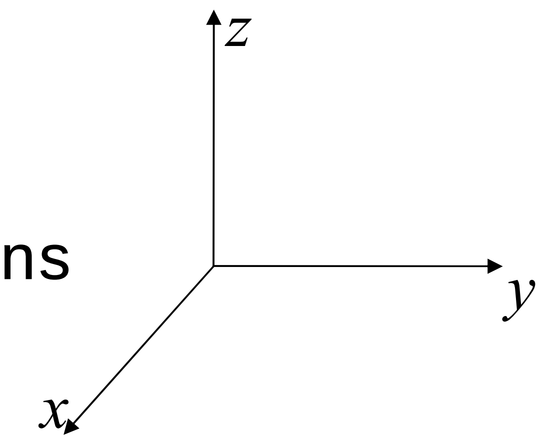
$$x' = a_{xx} x + a_{xy} y + b_x$$
$$y' = a_{yx} x + a_{yy} y + b_y$$

- Translation, rotation, scaling, reflection, shear. Any affine transformation can be expressed as the combination of these.
- Rotation, translation, reflection:
  preserve angles, lengths, parallel lines

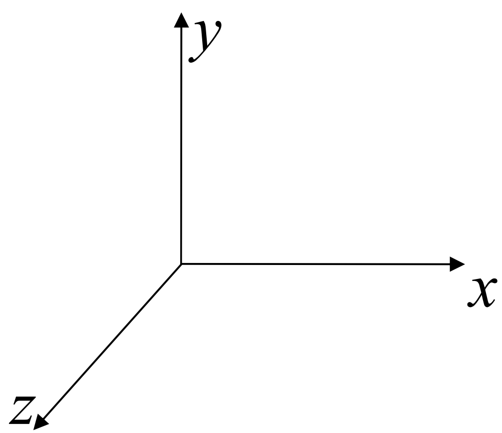
3 DIMENSIONAL TRANSFORMATIONS

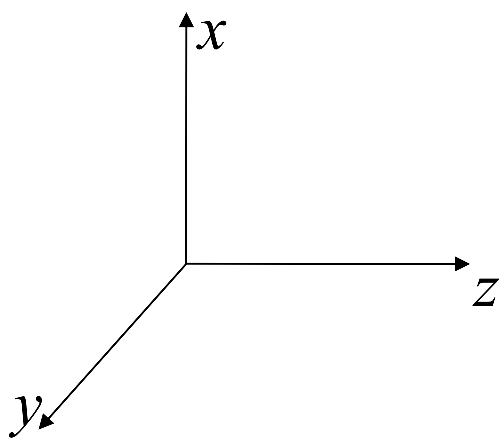
3D Transformations

- x,y,z coordinates. Usual notation: Right handed coordinate system



- Analogous to 2D we have 4 dimensions in homogenous coordinates.
- Basic transformations:
  - Translation
  - Rotation
  - Scaling



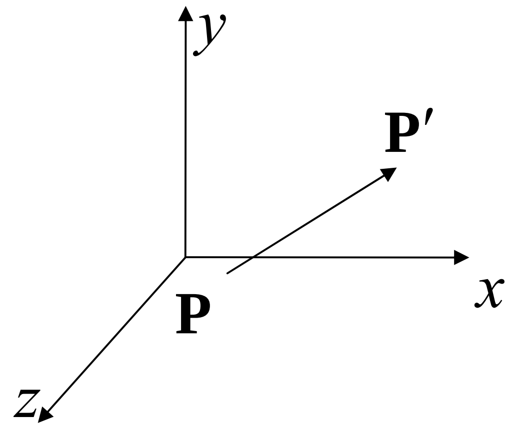


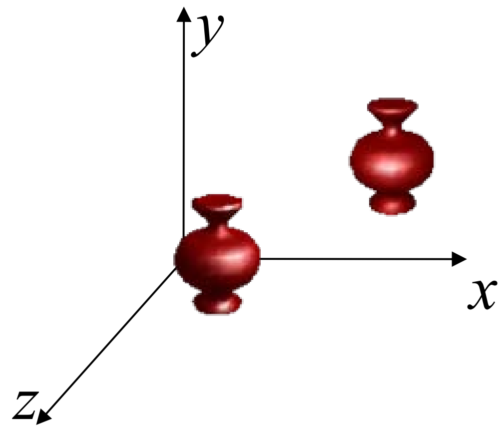
Translation

- move the object to a relative position.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

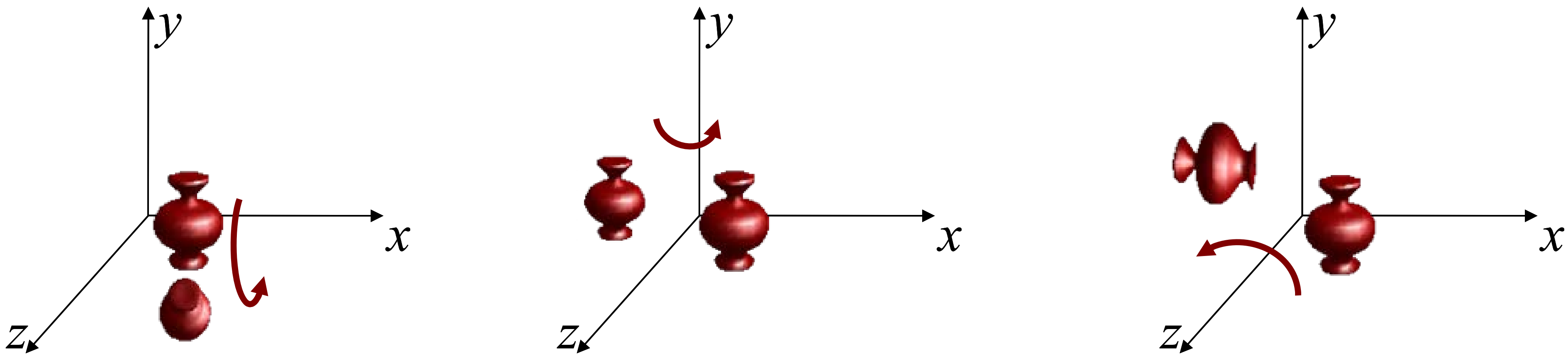
$$\mathbf{P}' = \mathbf{T} \cdot \mathbf{P}$$





Rotation

- Rotation arround the coordinate axes



x axis　　*y* axis　　*z* axis

Counterclockwise when looking along
the positive half towards origin

Rotation around coordinate axes

- Arround x

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

- Arround y

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

- Arround z

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

Rotation Arround a Parallel Axis

- Rotating the object around a line parallel to one of the axes: Translate to axis, rotate, translate back.

$$\mathbf{P}' = \mathbf{T}(0, y_p, z_p) \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T}(0, -y_p, -z_p) \cdot \mathbf{P}$$

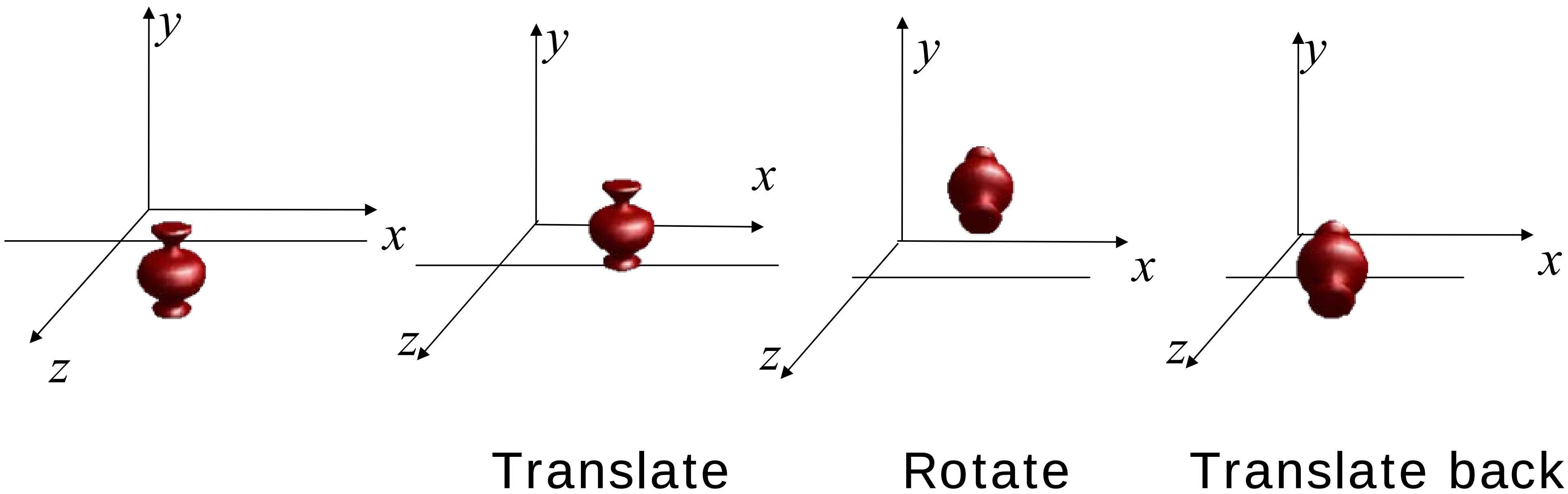


Figure from the textbook

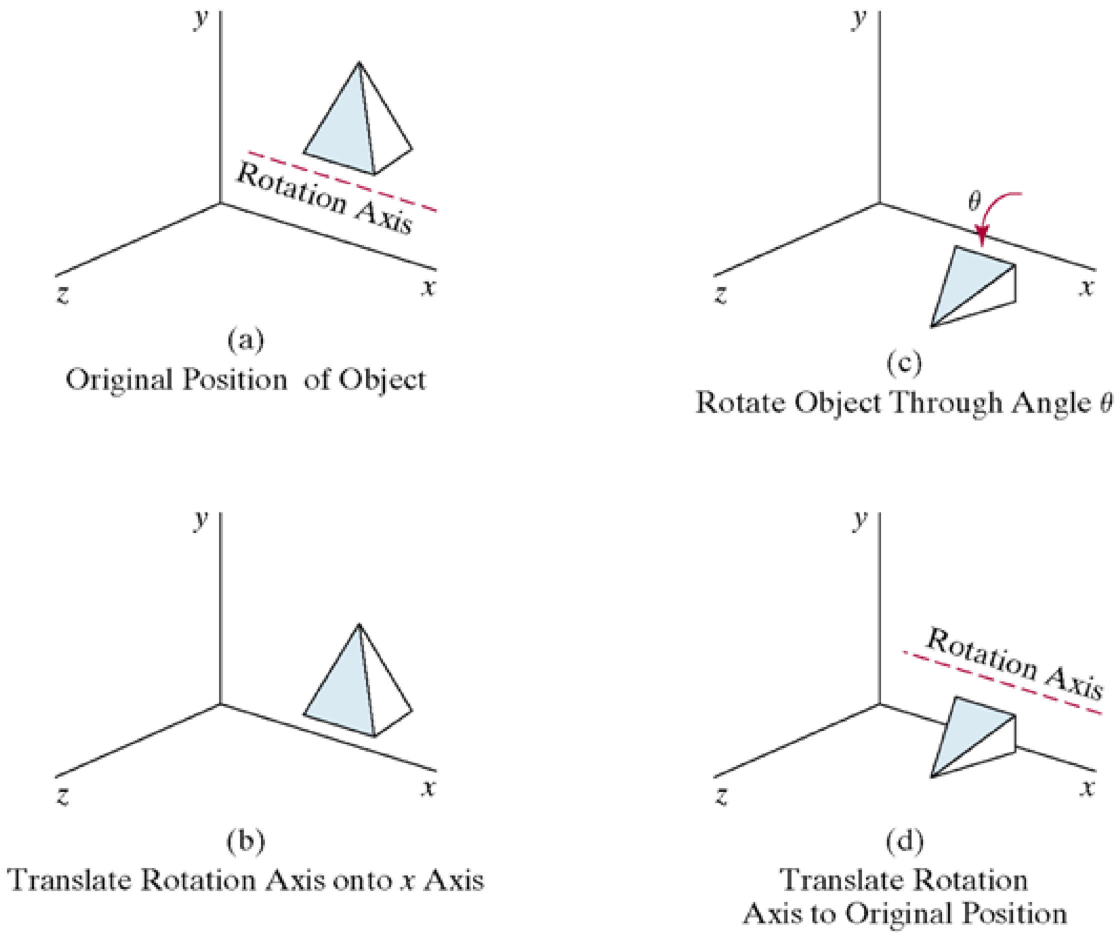
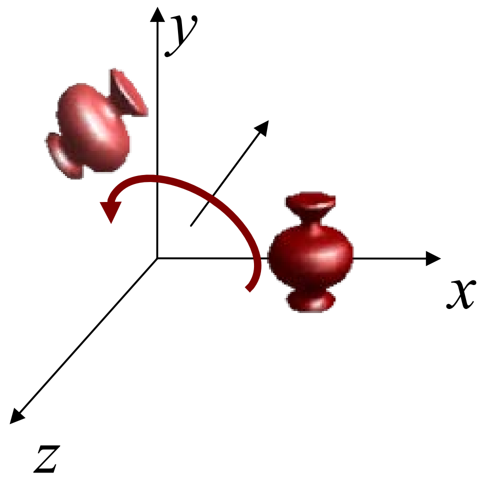


Figure 5-41

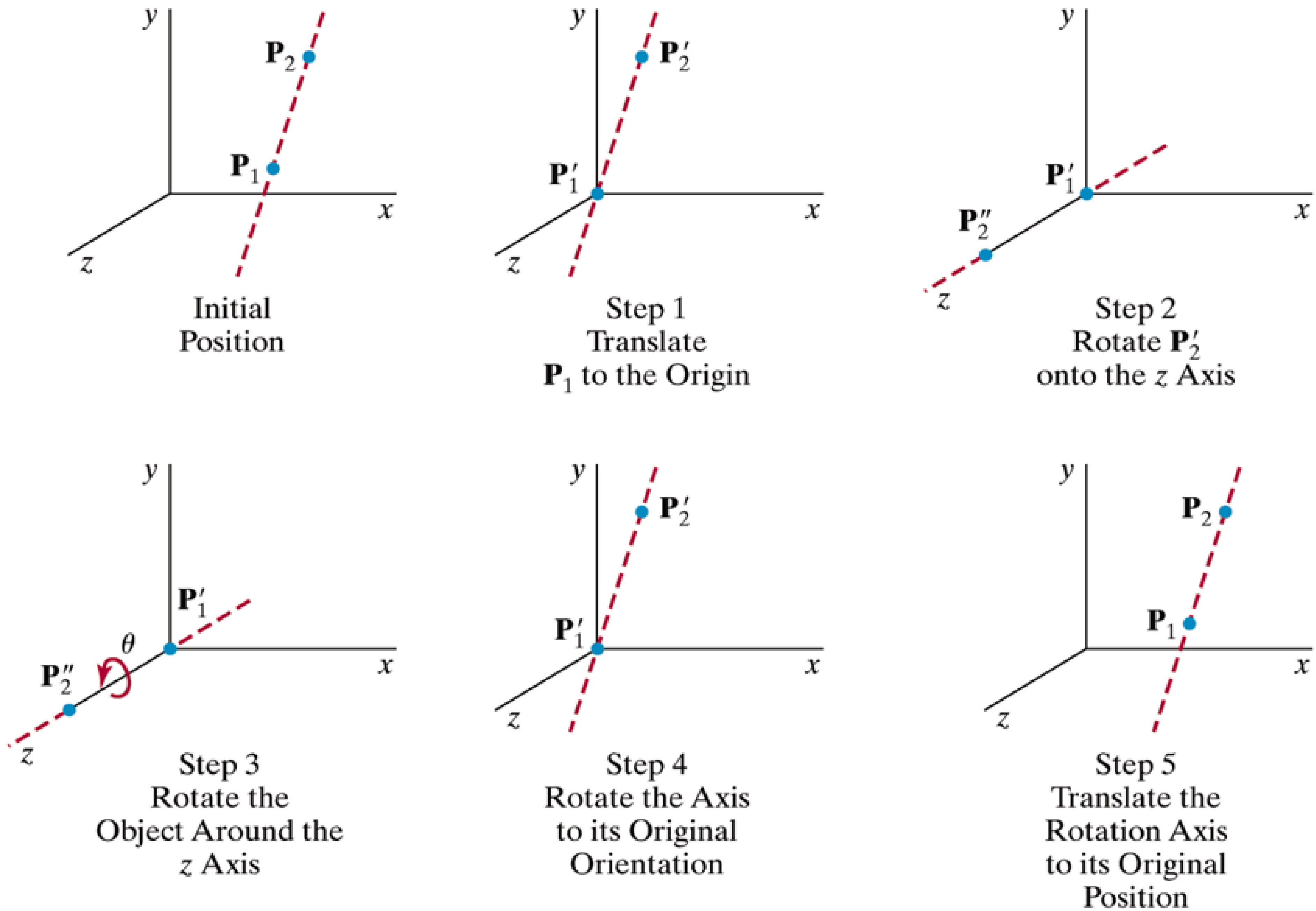
Sequence of transformations for rotating an object about an axis that is parallel to the x axis.

Rotation Around an Arbitrary Axis



- Translate the object so that the rotation axis passes though the origin
- Rotate the object so that the rotation axis is aligned with one of the coordinate axes
- Make the specified rotation
- Reverse the axis rotation
- Translate back

Rotation Around an Arbitrary Axis



Rotation Around an Arbitrary Axis

$$\mathbf{V} = \mathbf{P}_2 - \mathbf{P}_1 = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

$\mathbf{u}$ is the unit vector along $\mathbf{V}$: $\mathbf{u} = \dfrac{\mathbf{V}}{|\mathbf{V}|} = (a, b, c)$

First step: Translate $\mathbf{P}_1$ to origin:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

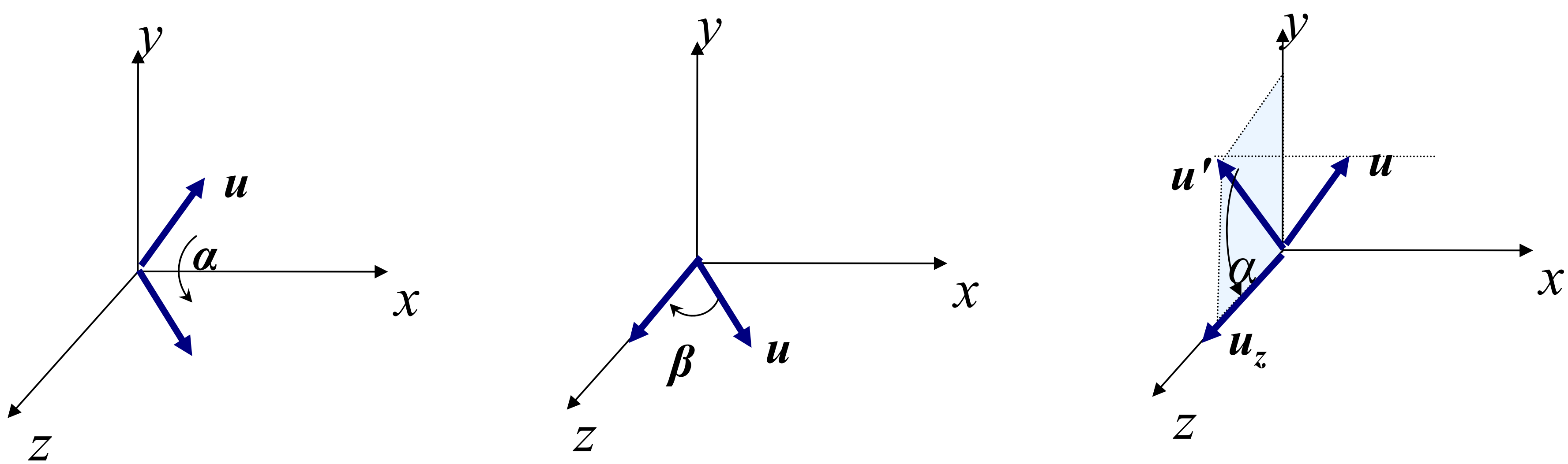
Next step: Align $\mathbf{u}$ with the z axis
we need two rotations: rotate around x axis to get $\mathbf{u}$ onto the xz plane, rotate around y axis to get $\mathbf{u}$ aligned with z axis.

Rotation Around an Arbitrary Axis

Align **u** with the z axis

1) rotate around x axis to get **u** into the xz plane,

2) rotate around y axis to get **u** aligned with the z axis



Dot product and Cross Product

- v dot u = vx * ux + vy * uy + vz * uz. That equals also to |v|*|u|*cos(a) if a is the angle between v and u vectors. Dot product is zero if vectors are perpendicular.

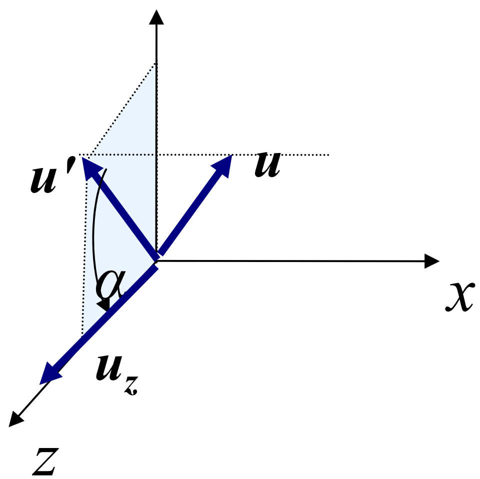
  v x u is a vector that is perpendicular to both vectors you multiply. Its length is |v|*|u|*sin(a), that is an area of parallelogram built on them. If v and u are parallel then the product is the null vector.

Rotation Around an Arbitrary Axis

Align $\mathbf{u}$ with the z axis

1) rotate around x axis to get $\mathbf{u}$ into the xz plane,

2) rotate around y axis to get $\mathbf{u}$ aligned with the z axis



$$\mathbf{u}' = (0, b, c)$$

Projection of $\mathbf{u}$ on yz plane

We need cosine and sine of α for rotation

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

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

$$b = d \sin\alpha$$

$$\cos\alpha = \frac{c}{d} \qquad \sin\alpha = \frac{b}{d}$$

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & -\frac{b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation Around an Arbitrary Axis

Align **u** with the z axis

1) rotate around x axis to get **u** into the xz plane,

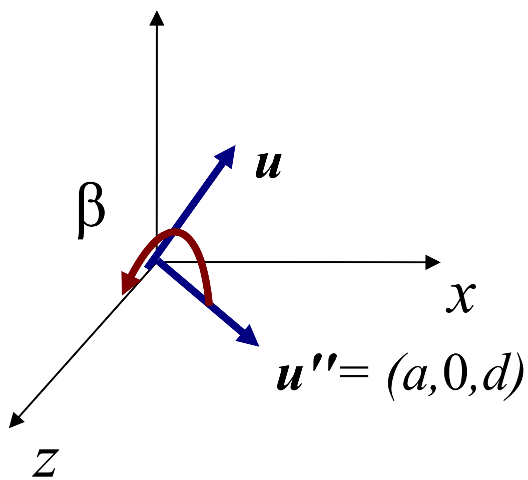
2) rotate around y axis to get **u** aligned with the z axis

$$\cos\beta = \frac{u'' \cdot u_z}{|u''| \cdot |u_z|} = d$$

$$\mathbf{u}'' \times \mathbf{u}_z = \mathbf{u}_y |\mathbf{u}''| |\mathbf{u}_z| \sin\beta = \mathbf{u}_y \cdot (-a)$$

$$\cos\beta = d \qquad \sin\beta = -a$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\mathbf{R}(\theta) = \mathbf{T}(x_1, y_1, z_1) \cdot \mathbf{R}_x(-\alpha) \cdot \mathbf{R}_y(-\beta) \cdot \mathbf{R}_z(\theta) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\alpha) \cdot \mathbf{T}(-x_1, -y_1, -z_1)$$

Rotation, … Alternative Method

Any rotation around origin can be represented by
3 orthogonal unit vectors:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This matrix can be thought of as rotating the unit r_{1*}, r_{2*}, and r_{3*} vectors onto x, y, and z axes.

So, to align a given rotation axis $\mathbf{u}$ onto the z axis,
we can define an (orthogonal) coordinate system and form this $\mathbf{R}$ matrix

Define a new coordinate system $(\mathbf{u}'_x, \mathbf{u}'_y, \mathbf{u}'_z)$
with the given rotation axis $\mathbf{u}$ using:

$$\mathbf{u}'_z = \mathbf{u}$$

$$\mathbf{u}'_y = \frac{\mathbf{u} \times \mathbf{u}_x}{|\mathbf{u} \times \mathbf{u}_x|}$$

$$\mathbf{u}'_x = \mathbf{u}'_y \times \mathbf{u}'_z$$

Rotation, … Alternative Method

$$\mathbf{u}'_z = \mathbf{u} = (a, b, c)$$

$$\mathbf{u}'_y = \frac{\mathbf{u} \times \mathbf{u}_x}{|\mathbf{u} \times \mathbf{u}_x|} = \frac{(a, b, c) \times (1,0,0)}{|\mathbf{u} \times \mathbf{u}_x|} = \frac{(0, c, -b)}{\sqrt{b^2 + c^2}} = (0, c/d, -b/d)$$

$$\mathbf{u}'_x = \mathbf{u}'_y \times \mathbf{u}'_z = (0, c/d, -b/d) \times (a, b, c) = (d, -a \cdot b/d, -a \cdot c/d)$$

$$\mathbf{R} = \begin{bmatrix} d & \frac{-a \cdot b}{d} & \frac{-a \cdot c}{d} & 0 \\ 0 & \frac{c}{d} & \frac{-b}{d} & 0 \\ a & b & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Check if this is equal to

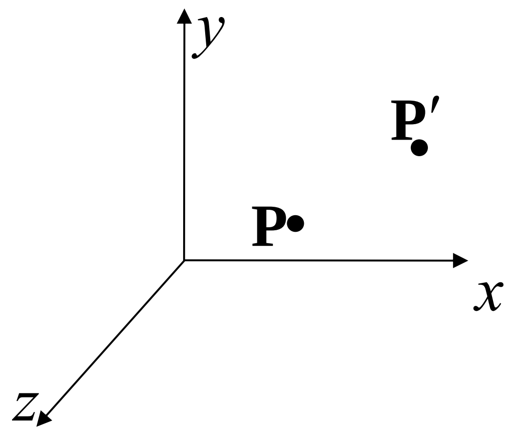
$$R_y(\beta) \cdot R_x(\alpha)$$

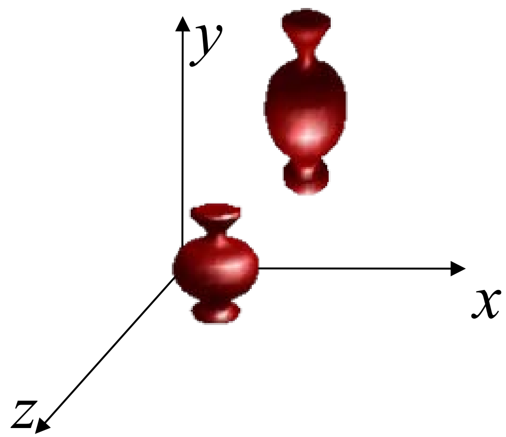
Scaling

- Change the coordinates of the object by scaling factors.

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

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$

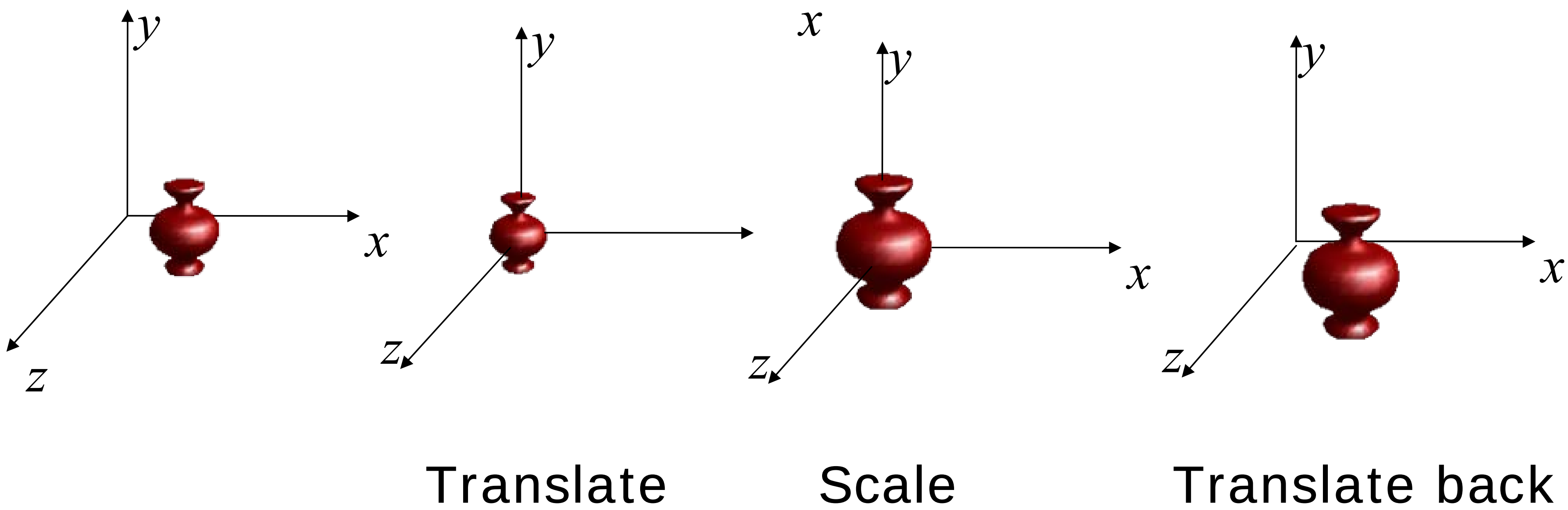




Scaling with respect to a Fixed Point

- Translate to origin, scale, translate back

$$\mathbf{P}' = \mathbf{T}(x_f, y_f, z_f) \cdot \mathbf{S} \cdot \mathbf{T}(-x_f, -y_f, -z_f) \cdot \mathbf{P}$$



Scaling with respect to a Fixed Point

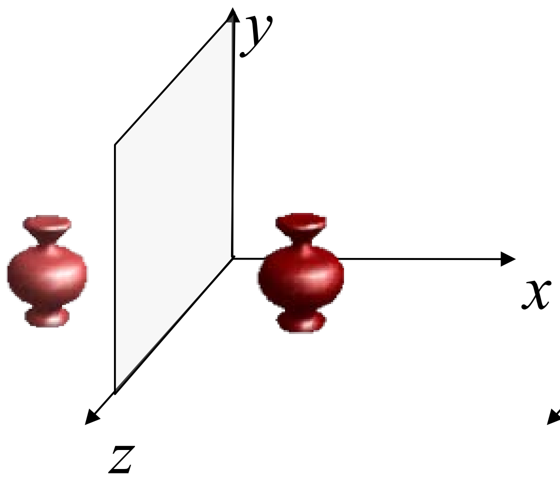
$$\mathbf{T}(x_f, y_f, z_f)\cdot\mathbf{S}\cdot\mathbf{T}(-x_f,-y_f,-z_f) = \begin{bmatrix} 1 & 0 & 0 & x_f \\ 0 & 1 & 0 & y_f \\ 0 & 0 & 1 & z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & x_f \\ 0 & 1 & 0 & y_f \\ 0 & 0 & 1 & z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & x_f \\ 0 & 1 & 0 & y_f \\ 0 & 0 & 1 & z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 & -s_x x_f \\ 0 & s_y & 0 & -s_y y_f \\ 0 & 0 & s_z & -s_z z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

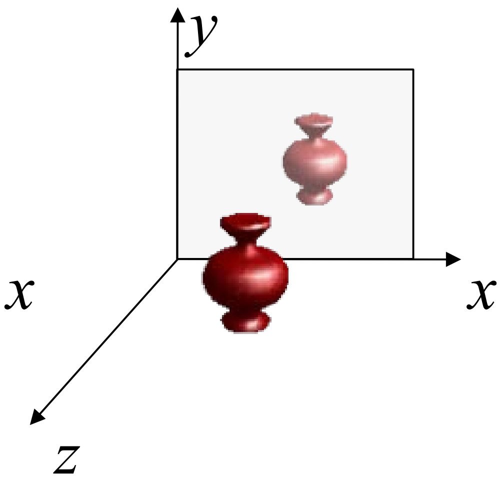
$$= \mathbf{T}(x_f, y_f, z_f)\cdot\mathbf{S}\cdot\mathbf{T}(-x_f,-y_f,-z_f) = \begin{bmatrix} s_x & 0 & 0 & x_f(1-s_x) \\ 0 & s_y & 0 & y_f(1-s_y) \\ 0 & 0 & s_z & z_f(1-s_z) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection

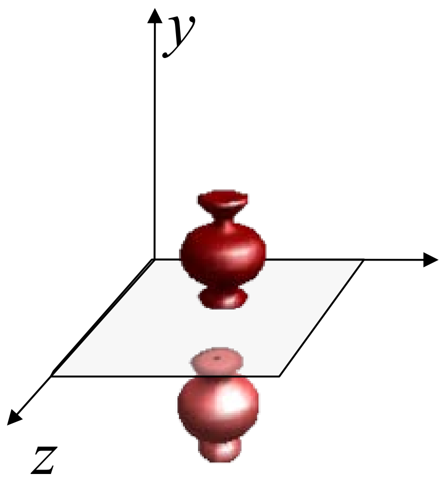
- Reflection over planes, lines or points



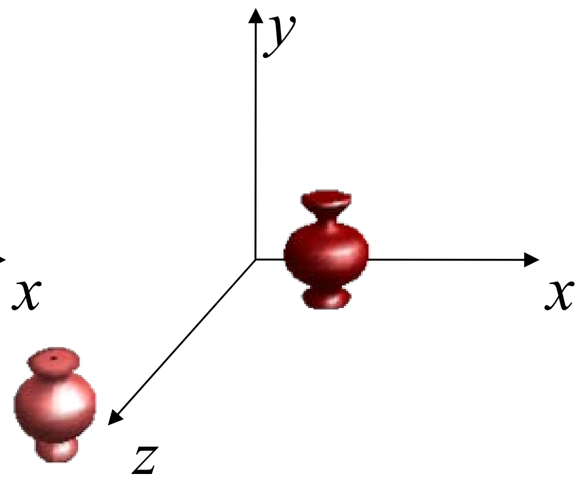
$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shear

- Deform the shape depending on another dimension

$$SH_z = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

x and y value depends on z value of the shape

$$SH_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ b & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

y and z value depends on x value of the shape

OpenGL Geometric-Transformation Functions

- In the core OpenGL library,
  - a separate function is available for each basic transformation (translate, rotate, scale)
  - all transformations are specified in 3D
- Parameters
  - Translation: translation amount in x, y, z axes
  - Rotation: angle, orientation of the rotation axis that passes through the origin
  - Scaling: scaling factors for three coordinates

Basic OpenGL Transformations

- glTranslate* (tx, ty, tz);
  - For 2D applications set tz = 0
- glRotate* (theta, vx, vy, vz);
  - theta in degrees
  - The rotation axis is defined by the vector (vx,vy,vz), i.e., P0 = (0,0,0) P1 = (vx,vy,vz)
- glScale* (sx, sy, sz);
  - Use negative values to get reflection transformation

OpenGL Matrix Operations

- glMatrixMode (GL_MODELVIEW);
  - *modelview mode* to tell OpenGL that we will be specifying geometric transformations. The command simply says that the current matrix operations will be applied on the 4 by 4 modelview matrix.
  - the other mode is the *projection mode*, which specifies the matrix that is used of projection transformations (i.e., how a scene is projected onto the screen)
  - There are also *color* and *texture* modes that we will discuss later

OpenGL Matrix Operations

- Once you are in the modelview mode, a call to a transformation routine generates a matrix that is multiplied by the current matrix for that mode
- **Whatever object defined is multiplied with the current matrix**
- The contents of the current matrix can also be manipulated explicitly
  - glLoadIdentity();
  - glLoadMatrix* (elements16);

  where elements16 is a single subscripted array that specifies a matrix in column-major order

OpenGL Matrix Operations

- Example:

```
for (int k=0; k<16;k++)
        elements16[k]=(float)k;
glLoadMatrixf(elements16);
```

will produce the matrix

$$\mathbf{M} = \begin{bmatrix} 0.0 & 4.0 & 8.0 & 12.0 \\ 1.0 & 5.0 & 9.0 & 13.0 \\ 2.0 & 6.0 & 10.0 & 14.0 \\ 3.0 & 7.0 & 11.0 & 15.0 \end{bmatrix}$$

OpenGL Matrix composition

- glMultMatrix* (otherElements16)
  - The current matrix is **postmultiplied** with the matrix specified in otherElements16

$$\mathbf{M}_{curr} = \mathbf{M}_{curr} \cdot \mathbf{M}'$$

what does this imply?

In a sequence of transformation commands, the last one specified in the code will be the first transformation to be applied.

OpenGL Matrix Stacks

- OpenGL maintains a matrix stack for all the four matrix modes
- When we apply geometric transformations using OpenGL functions, the 4 by 4 matrix at the top of the matrix stack is modified
- The top is also called the **current** matrix
- If we want to create multiple transformation sequences and save the composition results we can make use of the OpenGL matrix stack

OpenGL Matrix Stacks

- Initially, there is only the identity matrix in the stack
- To find out how many matrices are currently in the stack:
  - glGetIntegerv(GL_MODELVIEW_STACK_DEPTH,numMats)
- glPushMatrix ();
  - The current matrix is copied and stored in the second stack position
- glPopMatrix ();
  - Destroys the matrix at the top and the second matrix in the stack becomes the current matrix