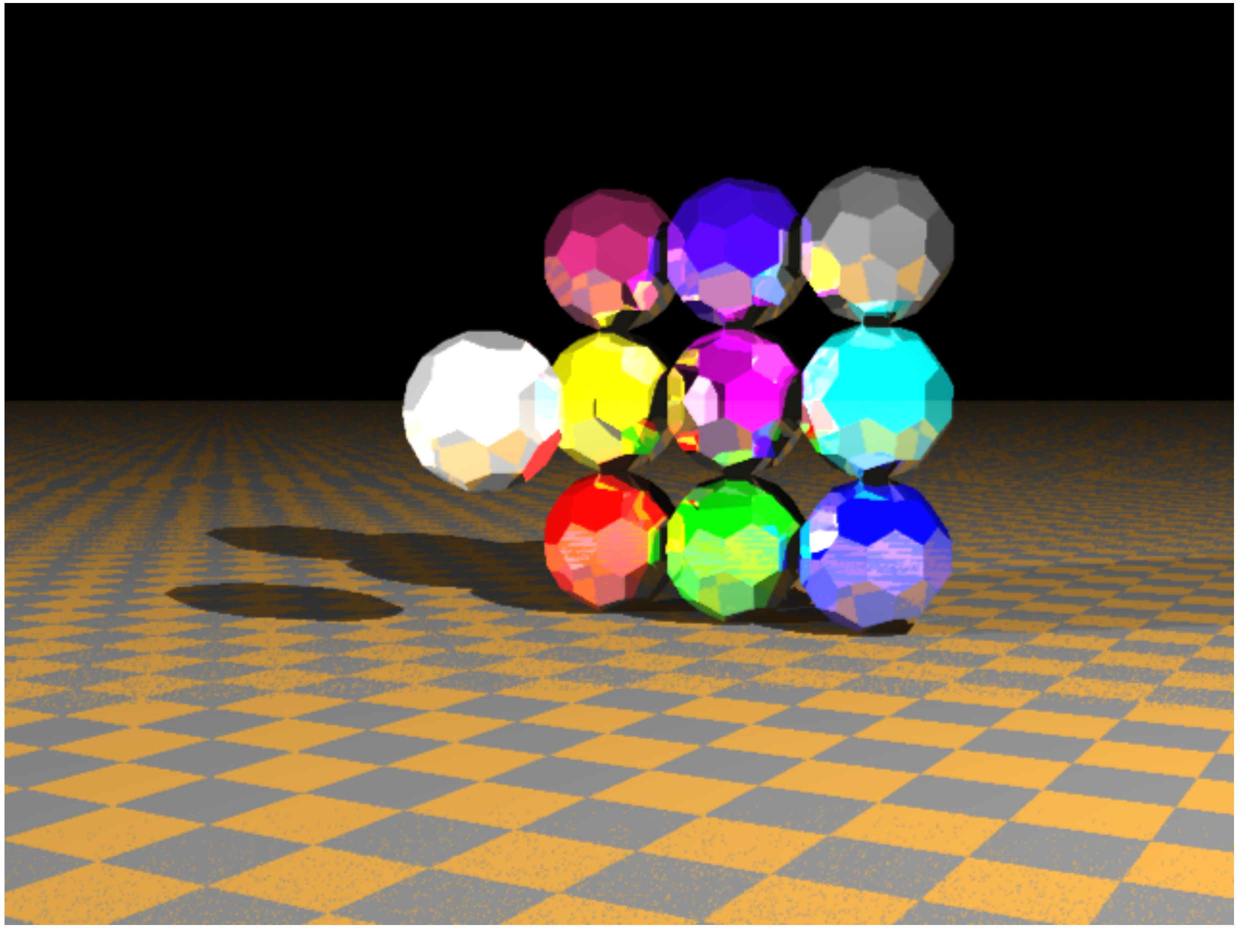
3D Geometric Transformations 3D



3D Translation

– Simple extension of the 2D translation

– Transformation T defined in 4D homogeneous coordinate system

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

– Or $P' = T \cdot P$

3D Rotation

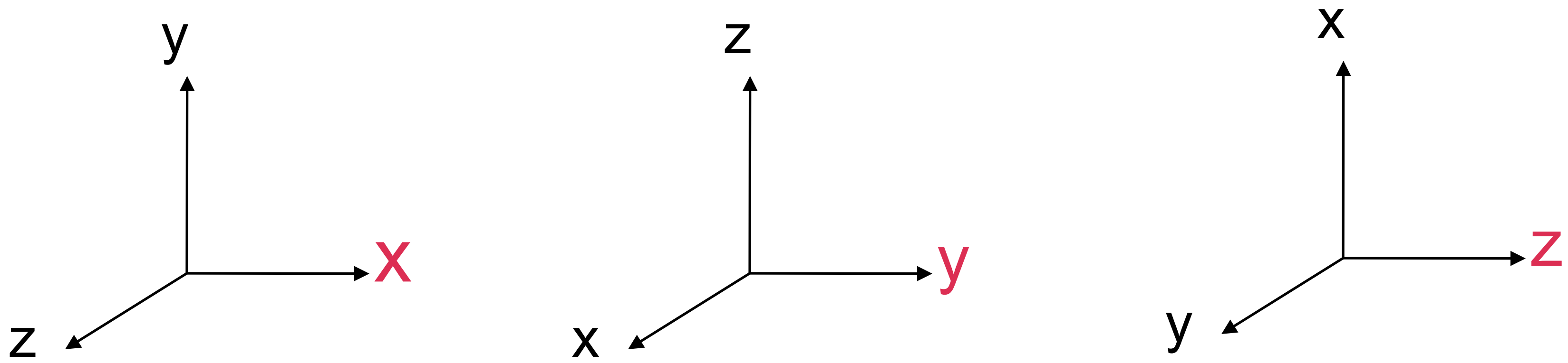
- Extend 2D by defining rotation about the three principal axis
- For rotations about the z-axis

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

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

- Or $P' = R_z(\theta) \cdot P$

– For rotations about x and y axes note the cyclic permutations ($x \rightarrow y \rightarrow z \rightarrow x$) present in the following diagrams



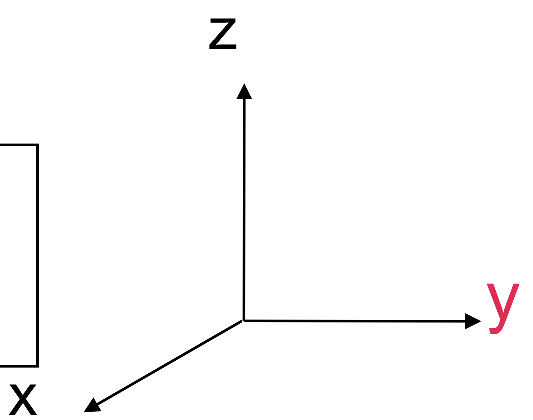
- x and y rotation matrices can be derived from previous slide by replacing ($x \rightarrow y \rightarrow z \rightarrow x$)

– Rotation about x-axis

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

Rotation about Z:

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

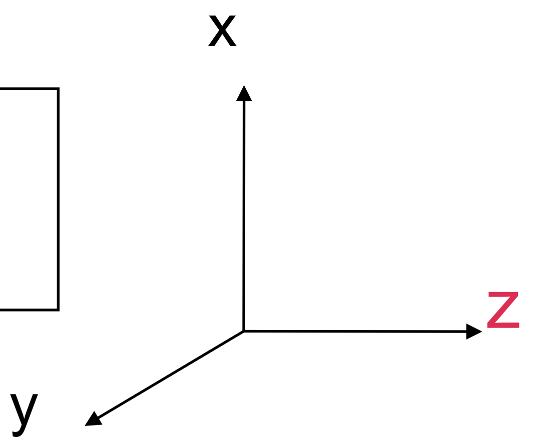


$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

– Rotation about y-axis

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

Rotation about X:
$(\ y' = y\cos\theta - z\sin\theta)$
$(\ z' = y\sin\theta + z\cos\theta)$



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

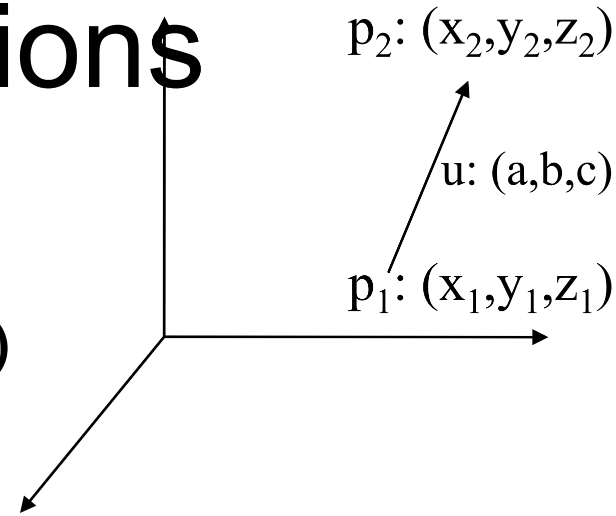
– Inverse of a rotation gotten by replacing θ by –θ

– Inverse of a rotation matrix is its transpose

- only sine function affected by the change in sign
- sine function occurs on the off diagonal and is symmetric

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

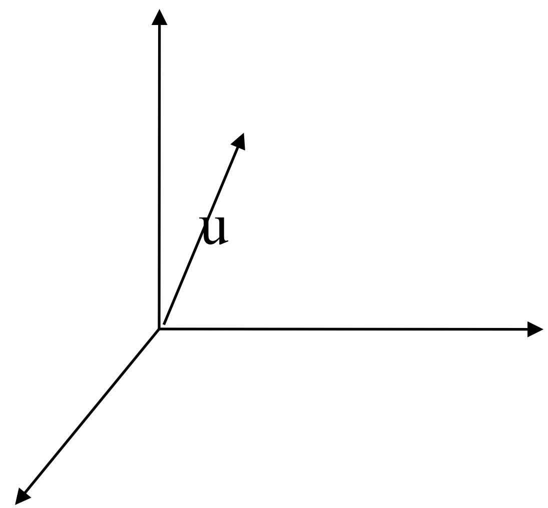
General 3D rotations



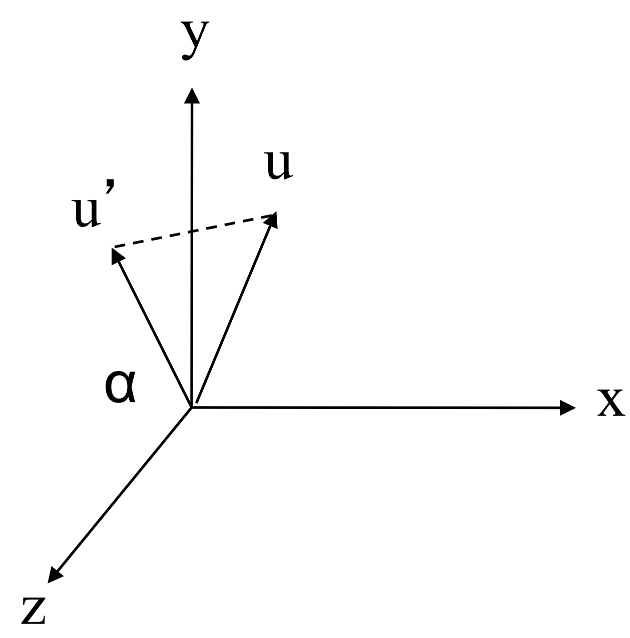
– Strategy
  (rotate about unit rotation vector u)
  - Translate one of the points to the origin
  - Rotate so that the vector is along one of the axes
  - Perform the rotation
  - Rotate back
  - Translate back

– Translate axis of rotation to origin

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



- Rotate about x so that u = (a, b, c) lies on the x-z plane
  - need cosine and sine of α for matrix
  - u' is projection of u onto the y-z plane: u' = (0, b, c)
  - x=(1,0,0), z=(0,0,1)
  - dot product of u' with the unit vector z
  - Equate algebraic definition of cross product: with parallelogram area definition:

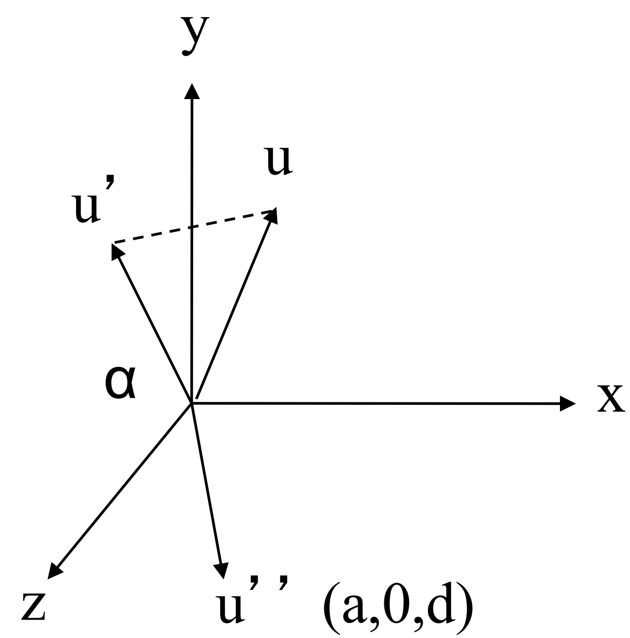


$$\boxed{\cos\alpha = \frac{\mathbf{u}'\bullet\mathbf{z}}{|\mathbf{u}'|} = \frac{c}{\sqrt{b^2+c^2}} \equiv \frac{c}{d}}$$

$$\mathbf{u}'\times\mathbf{z} = \mathbf{x}|\mathbf{u}'|\sin\alpha$$

$$\mathbf{u}'\times\mathbf{z} = \mathbf{x}b*1 \quad (height*base)$$

$$\boxed{\sin\alpha = \frac{b}{d}}$$



- therefore, the rotation about the x axis so that u lies in the x-z plane is:

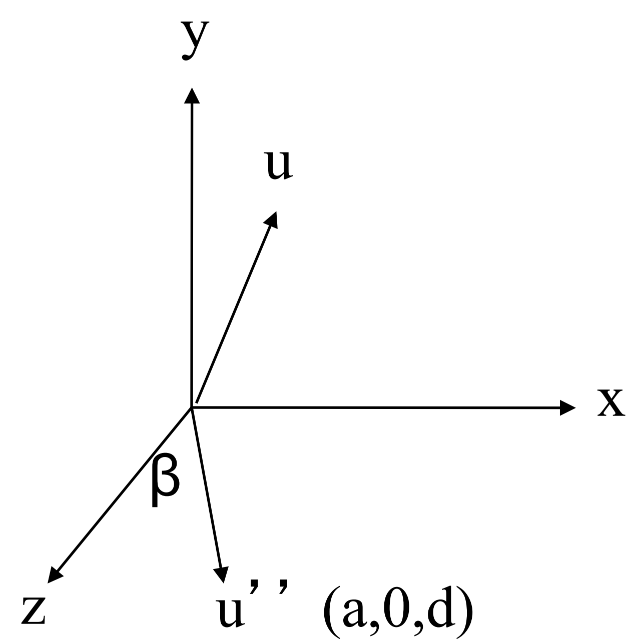
$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & -\frac{b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\cos\alpha = \frac{c}{d}$$

$$\sin\alpha = \frac{b}{d}$$

- rotation about x leaves x component (a) unchanged and z component is length of u' ($d=\mathrm{sqrt}(b^2+c^2)$)

– Rotate β about y axis so that u'' lie on the z axis



- again, need sine and cosine of β
- dot product of u'' with the unit vector z

  (|u''| = |u| =1.0)
- cross product of u'' with z is:
- which is also equal to:

- equate equations get sine:

$$\boxed{\cos\beta = \frac{\mathbf{u}'' \bullet \mathbf{z}}{|\mathbf{u}''|} = d}$$

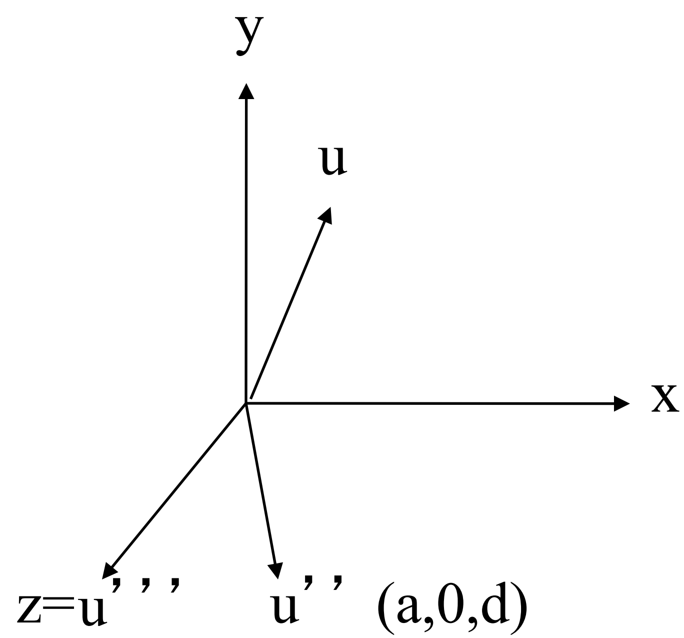
$$\mathbf{u}'' \times \mathbf{z} = \mathbf{y}|\mathbf{u}''|\sin\beta$$

$$\mathbf{u}'' \times \mathbf{z} = \mathbf{y}(-a)$$

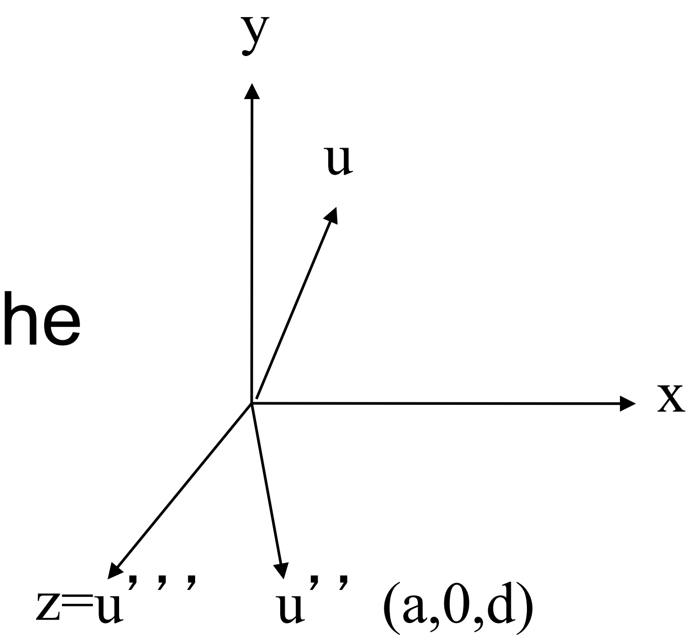
$$\boxed{\sin\beta = -a}$$

- therefore, the rotation about the y axis so that u'' lie along the z axis is

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



– Now rotate about the z axis by the desired amount of rotation θ



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

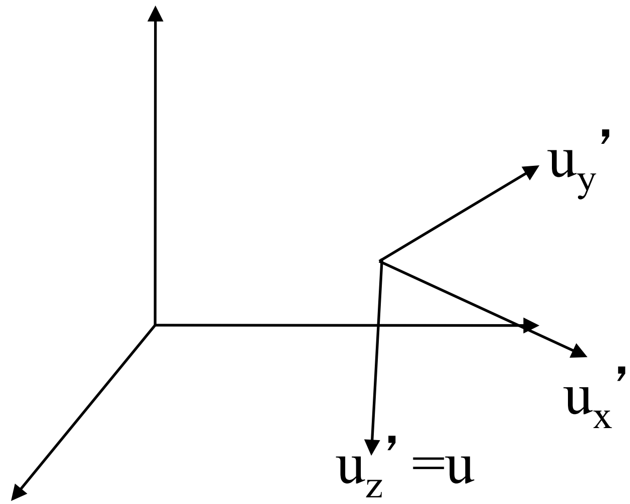
– Do inverse rotations and translation to get back to the starting postion.

– Transformation matrix to rotate about an arbitrary axis

$$\boxed{R(\theta) = T^{-1} R_x^{-1}(\alpha) R_y^{-1}(\beta) R_z(\theta) R_y(\beta) R_x(\alpha) T}$$

– Rotation matrix $R_y(\beta)R_x(\alpha)$ could have been derived by noting that rotation matrices are orthogonal

- rows and columns are orthonormal
- form new orthogonal axis



$$u_z^{'} = u \quad u_y^{'} = \frac{u \times x}{|u \times x|} \quad u_x^{'} = u_y^{'} \times u_z^{'}$$

- and use their components as the rows of the rotation matrix
- i.e. you want a matrix that will rotate the primed into the unprimed coordinate system

3D Scale

– Simple extension of 2D scale

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

– The lower right matrix component can be used to do a uniform scale

– Scale about a specific point can be accomplished by first translating that point to the origin then scaling

3D Reflections

– Extension of 2D reflections
– One usage is to change from right-handed coordinate system to the left-handed coordinate system (common in computer graphics)

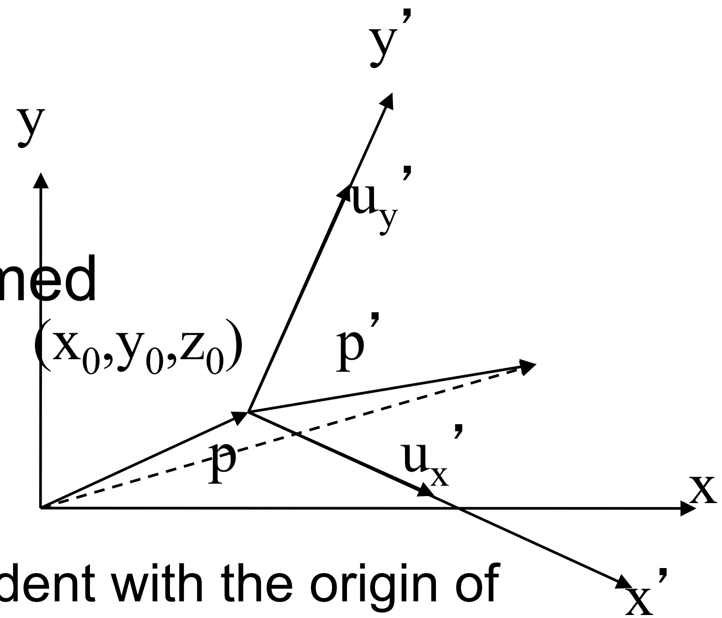
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Shear

– z-axis shear relative to a reference position
  - leave z axis unchanged and move x and y by an amount proportional to $z - z_{ref}$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & sh_{zx} & -sh_{zx} \cdot z_{ref} \\ 0 & 1 & sh_{zy} & -sh_{zy} \cdot z_{ref} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Transformations between 3D coordinate systems



- Transformation matrix that will transform a vector p from the unprimed to the primed coordinate system is concatenation of
  - Translation that will make the origin of the primed coordinate system coincident with the origin of the unprimed coordinate system
    $T(-x_0,-y_0,-z_0)$
  - Rotation that will make the axes of the primed coordinate system coincident with the unprimed coordinate system

$$R = \begin{bmatrix} u'_{x1} & u'_{x2} & u'_{x3} & 0 \\ u'_{y1} & u'_{y2} & u'_{y3} & 0 \\ u'_{z1} & u'_{z2} & u'_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Affine transformations

- Transformed coordinates are a linear function of the untransformed coordinates

  $x' = a_{xx}\, x + a_{xy}\, y + a_{xz}\, z + b_x$

  $y' = a_{yx}\, x + a_{yy}\, y + a_{yz}\, z + b_y$

  $z' = a_{zx}\, x + a_{zy}\, y + a_{zz}\, z + b_z$

- Parallel lines transform to parallel lines
- Finite points map to finite points

- Examples: translation, rotation, scaling, reflection, shear
- Scaling and shear do not necessarily preserve angles and lengths
- Translation, rotation, reflection do preserve angles and lengths

Constructing transformation matrix in OpenGL

– translation matrix (4x4)

```
glTranslate*( tx, ty, tz );
```

- *: float or double

– rotation matrix

```
glRotate*( thetadeg, vx, vy, vz );
```

– scaling matrix

```
glScale*( sx, sy, sz );
```

– To set geometric transformation mode

```
glMatrixMode( GL_MODELVIEW );
```

– This sets a 4x4 modelview matrix as the current matrix
– Any subsequent transformation routines from previous slide multiplied by the current matrix
– To assign an identity matrix as the current matrix

```
glLoadIdentity( );
```

OpenGL Matrix Stack

– For each matrix mode set by glMatrixMode function (modelview, projection, texture, and color)
  - OpenGL maintains a matrix stack
  - Initially contains a single identity matrix
  - At any time, the top of each stack is the current matrix for that mode

- To load an arbitrary 4x4 matrix as the current matrix

```
glLoadMatrix*( elements16 );
```

- To multiply current matrix by an arbitrary matrix:

```
glMultMatrix*( elements16 );
```

  - current matrix is post multiplied by the specified matrix and the result becomes the new current matrix
    e.g. if M is the current matrix, the operation with M' as the argument will put $\mathrm{M} \cdot \mathrm{M}'$ as the current modelview matrix
  - can have any number of these multiplication operations
  - the last matrix specified is the first one that is applied

– We can copy the current top of the stack onto the second stack position

```
glPushMatrix*( );
```

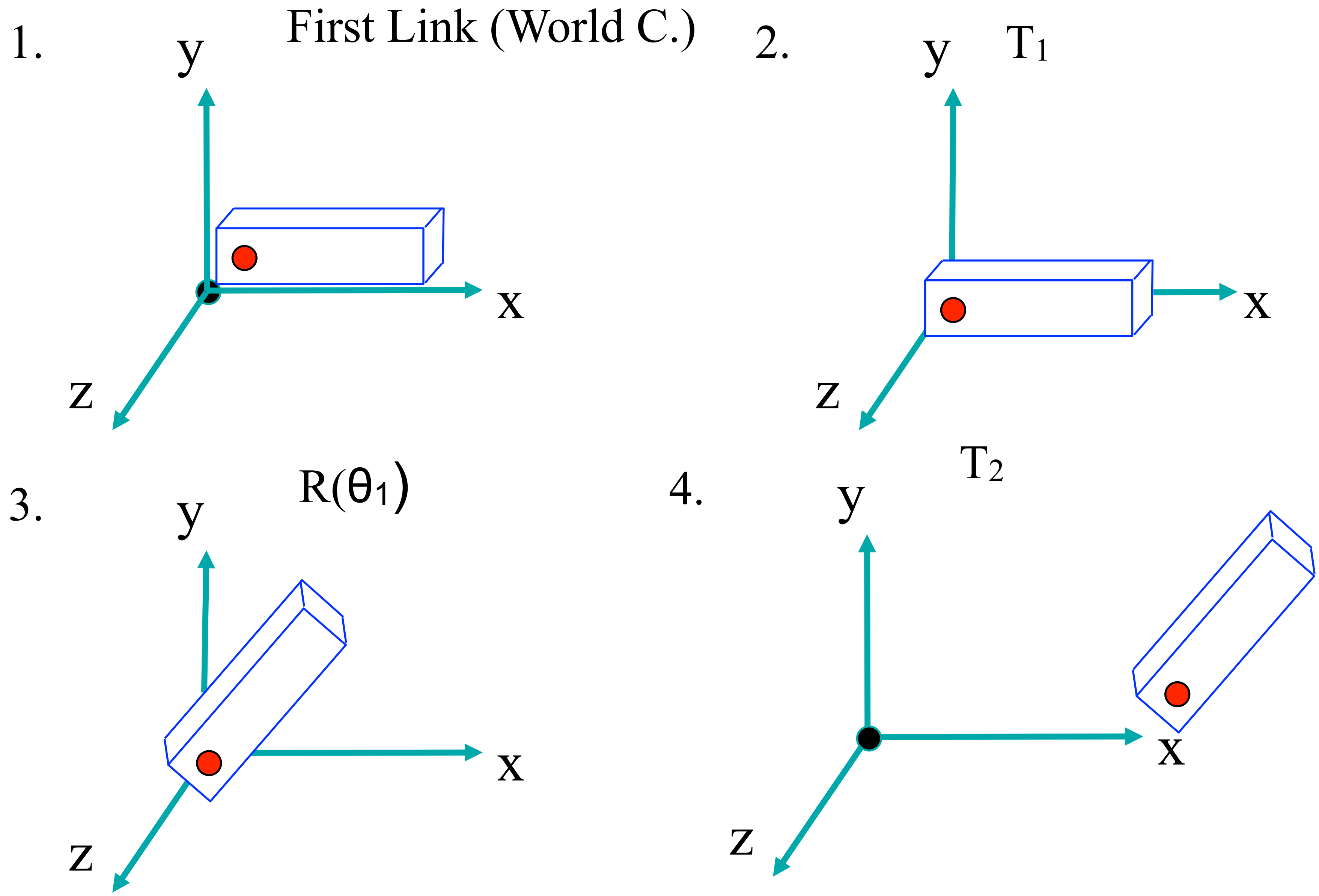
- gives duplicate matrix on top two positions on the stack
- allow storing of a composite matrix for later re-use

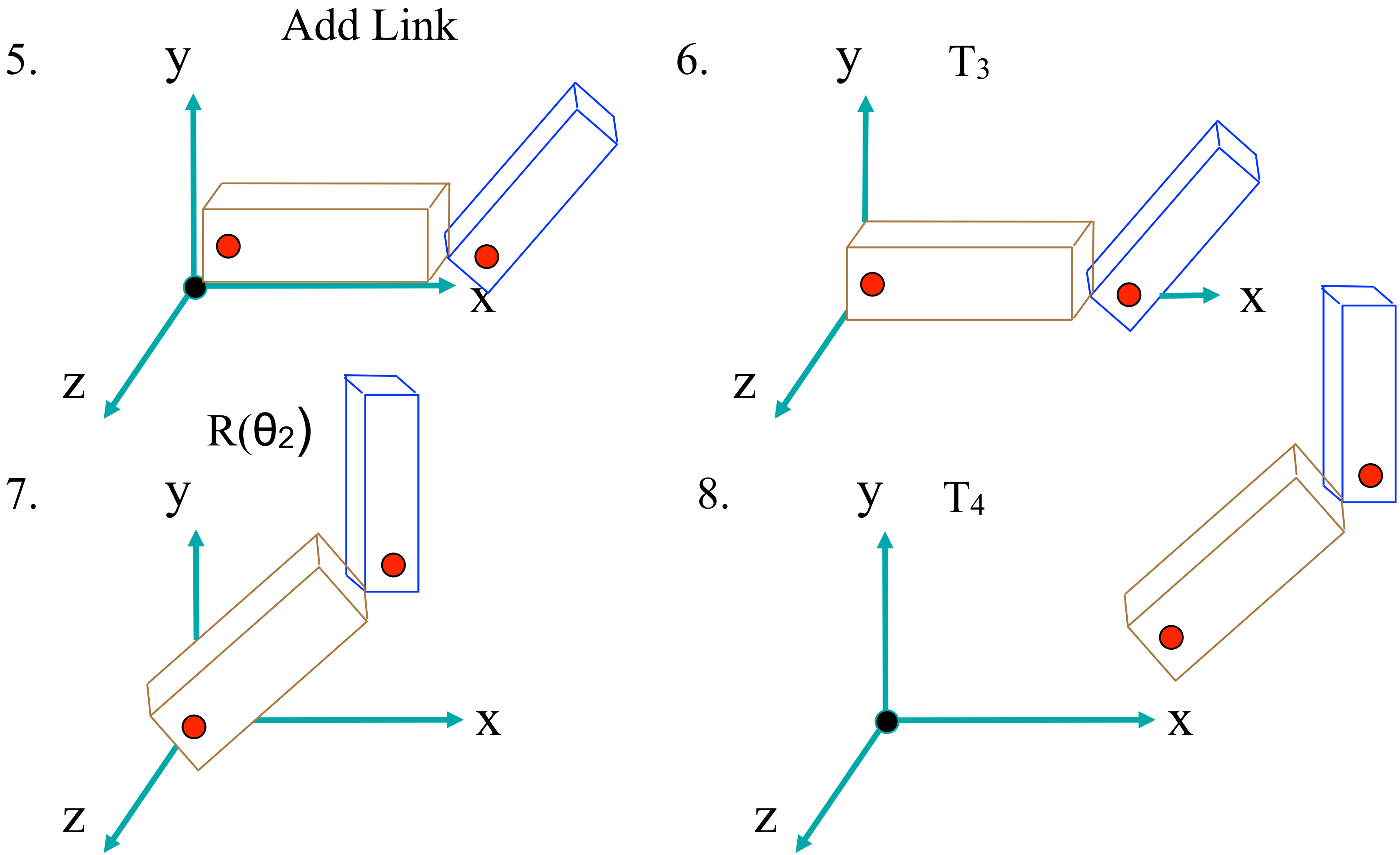
– To pop off the top of the stack

```
glPopMatrix*( );
```

Hierarchical Modeling

- More complex objects composed of simpler ones
- Transformations connect objects
- Links keep parts together
- When you move a shoulder, the forearm should move with it.
- Important for animation


1.
First Link (World C.)
y
x
z
2.
y
T_1
x
z
3.
y
$R(\theta_1)$
x
z
4.
T_2
y
x
z


5.
Add Link
y
x
z
6.
y
T3
x
z
7.
R(θ2)
y
x
z
8.
y
T4
x
z

Transformation hierarchy example

glLoadIdentity()	I
Transformation T1	T1
Draw something (T1)	
glPushMatrix()	T1 T1
Transformation T2	T1*T2 T1
glPushMatrix()	T1*T2 T1*T2 T1
Transformation T3	T1*T2*T3 T1*T2 T1
Draw something (T1*T2*T3)	
glLoadIdentity()	I T1*T2 T1
Draw something (I)	
glPopMatrix()	T1*T2 T1
Draw something (T1*T2)	
glPopMatrix()	T1
Draw something (T1)	

Next: 3D Viewing Transformations