CS 4204 Computer Graphics

3D Transformations

Yong Cao

Virginia Tech

References:

"Introduction to Computer Graphics" course notes by Petros Faloutsos, UCLA

Affine Transformations in 3D

General form

$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

Elementary 3D Affine Transformations

Translation

$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

Scaling Around the Origin

$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

Shear around the origin

Along x-axis

$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & a & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

3DRotation

Various representations

Decomposition into axis rotations (x-roll, y-roll, z-roll)

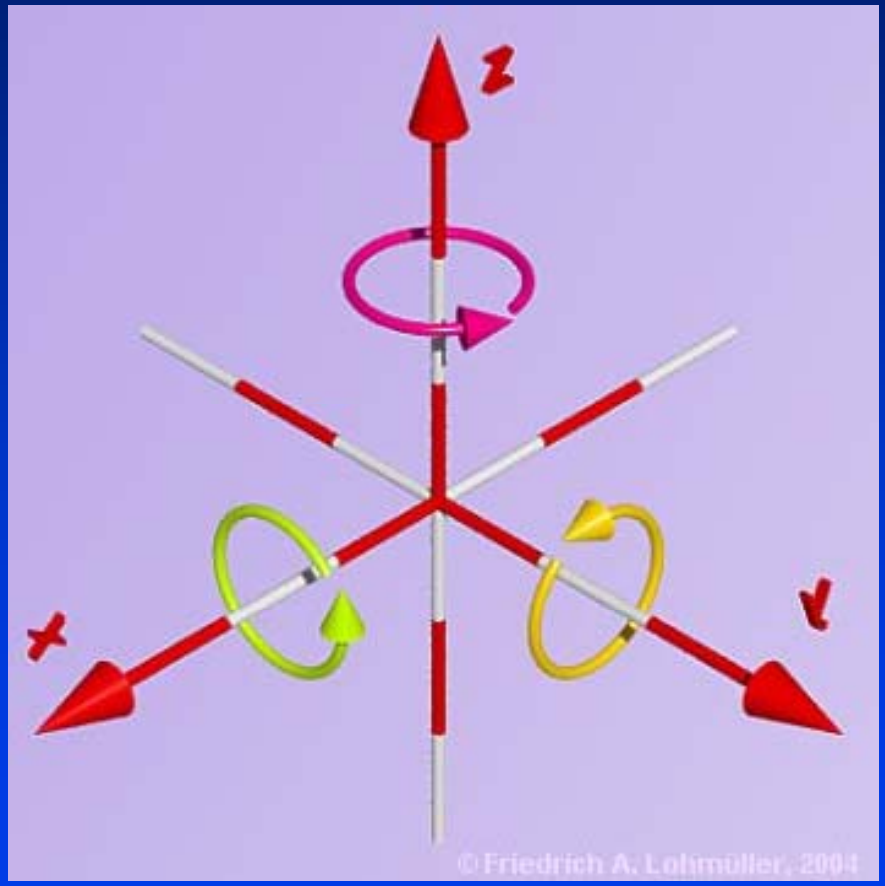
CCW positive assumption

Reminder 2D z-rotation

$$Q_x = cos\theta P_x - sin\theta P_y$$
$$Q_y = sin\theta P_x + cos\theta P_y$$

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}$$

Three axis to rotate around



Z-roll

$$Q_x = cos\theta P_x - sin\theta P_y$$
$$Q_y = sin\theta P_x + cos\theta P_y$$
$$Q_z = P_z$$

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

X-roll

Cyclic indexing

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ x \\ y \end{bmatrix}$$

$$x \rightarrow y \rightarrow z \rightarrow x \rightarrow y$$

$$Q_y = cos\theta P_y - sin\theta P_z$$
$$Q_z = sin\theta P_y + cos\theta P_z$$
$$Q_x = P_x$$

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

Y-roll

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ x \\ y \end{bmatrix}$$

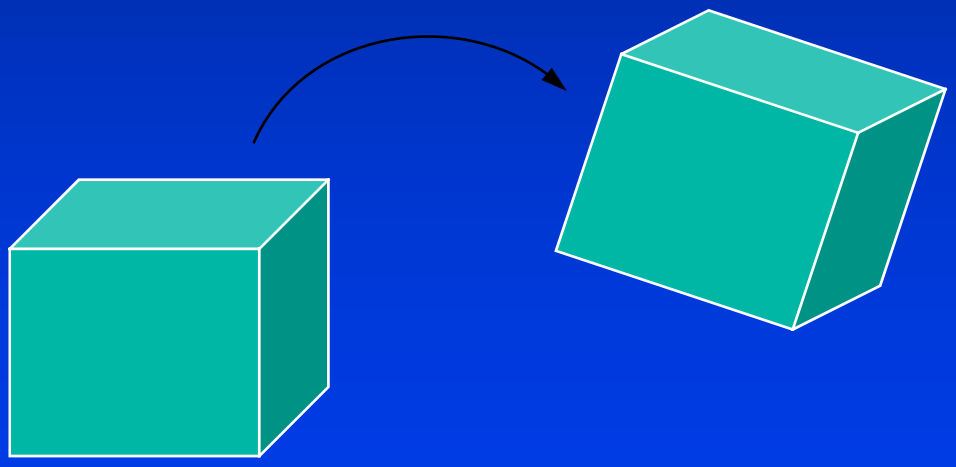
$$Q_z = cos\theta P_z - sin\theta P_x$$
$$Q_x = sin\theta P_z + cos\theta P_x$$
$$Q_y = P_y$$

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

Rigid body transformations

Translations and rotations

Preserve lines, angles and distances



Inversion of transformations

Translation: $T^{-1}(a,b,c) = T(-a,-b,-c)$

Rotation: $R^{-1}_{axis}(b) = R_{axis}(-b)$

Scaling: $S^{-1}(sx,sy,sz) = S(1/sx,1/sy,1/sz)$

Shearing: $Sh^{-1}(a) = Sh(-a)$

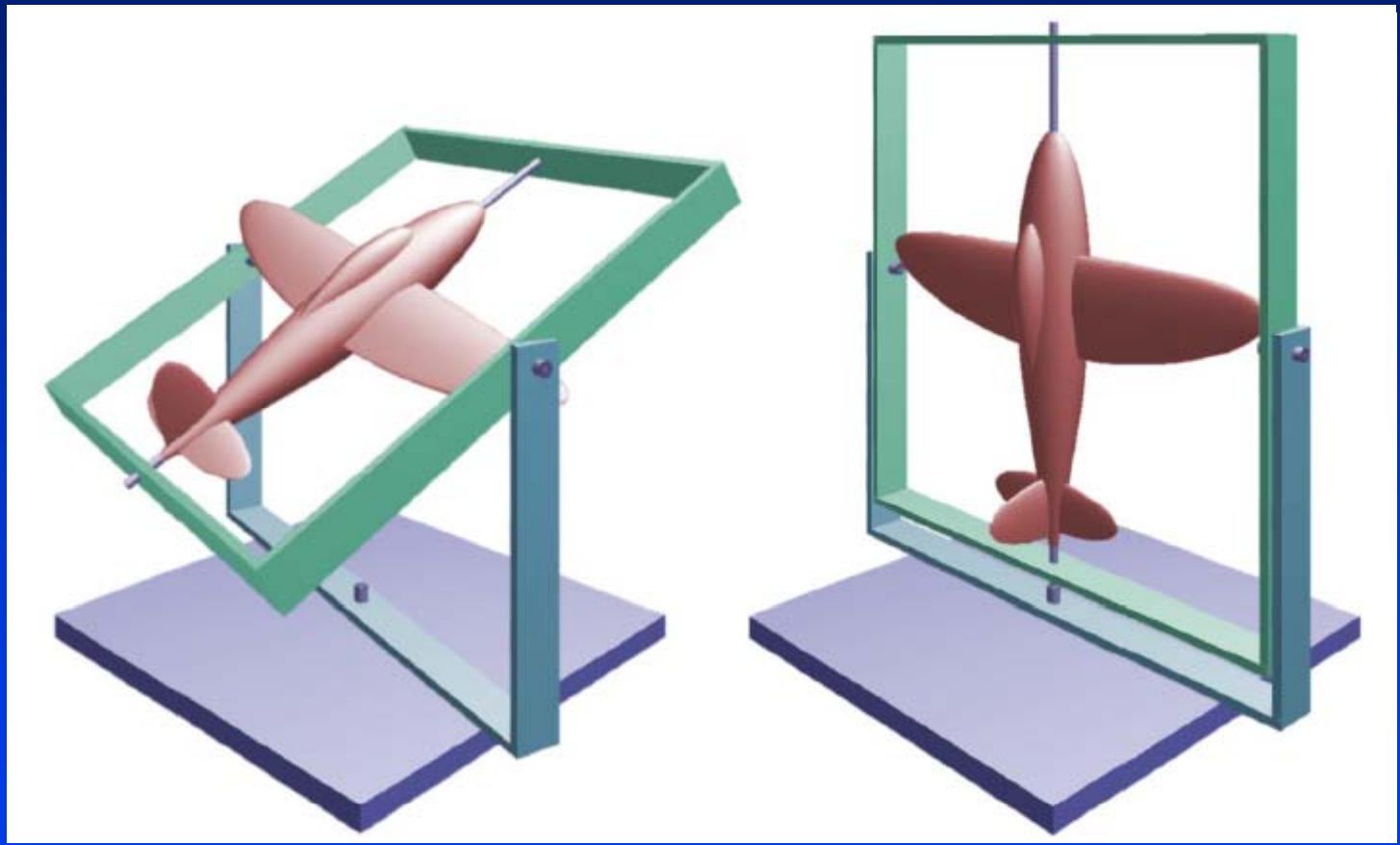
Inverse of Rotations

Pure rotation only, no scaling or shear.

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$$

$$M^{-1} = M^{T}$$

Gimball lock



Gimball lock

$$R(\theta_1, \theta_2, \theta_3) = R_z(\theta_3) R_y(\theta_2) R_x(\theta_1)$$

$$\begin{pmatrix} cos(\theta_3) & -sin(\theta_3) & 0 & 0 \\ sin(\theta_3) & cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos(\theta_2) & 0 & sin(\theta_2) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta_2) & 0 & cos(\theta_2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta_1) & -sin(\theta_1) & 0 \\ 0 & sin(\theta_1) & cos(\theta_1) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R(\theta_1, 90^o, \theta_3) = R_z(\theta_3) R_y(90^o) R_x(\theta_1)$$

$$\begin{pmatrix} cos(\theta_3) & -sin(\theta_3) & 0 & 0 \\ sin(\theta_3) & cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta_1) & -sin(\theta_1) & 0 \\ 0 & sin(\theta_1) & cos(\theta_1) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$\begin{pmatrix} cos(\theta_3) & -sin(\theta_3) & 0 & 0 \\ sin(\theta_3) & cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & sin(\theta_1) & cos(\theta_1) & 0 \\ 0 & cos\theta_1) & -sin(\theta_1) & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Loss of degree of freedom

$$R(\theta_1, 90^o, \theta_3) = \begin{pmatrix} cos(\theta_3) & -sin(\theta_3) & 0 & 0 \\ sin(\theta_3) & cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & sin(\theta_1) & cos(\theta_1) & 0 \\ 0 & cos\theta_1) & -sin(\theta_1) & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$\begin{pmatrix} 0 & cos(\theta_3)sin(\theta_1) - sin(\theta_3)cos(\theta_1) & cos(\theta_3)cos(\theta_1) + sin(\theta_3)sin(\theta_1) & 0 \\ 0 & cos(\theta_3)cos(\theta_1) + sin(\theta_3)sin(\theta_1) & -cos(\theta_3)sin(\theta_1) + sin(\theta_3)cos(\theta_1) & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$\begin{pmatrix} 0 & sin(\theta_1 - \theta_3) & cos(\theta_1 - \theta_3) & 0 \\ 0 & cos\theta_1 - \theta_3) & -sin(\theta_1 - \theta_3) & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$\begin{pmatrix} 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & cos\theta) & -sin(\theta) & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = R(\theta) \qquad (\theta_1, \theta_3) \rightarrow \theta = (\theta_1 - \theta_3)$$

Rotation around an arbitrary axis

Euler's theorem: Any rotation or sequence of rotations around a point is equivalent to a single rotation around an axis that passes through the point.

What does the matrix look like?

Rotation around an arbitrary axis

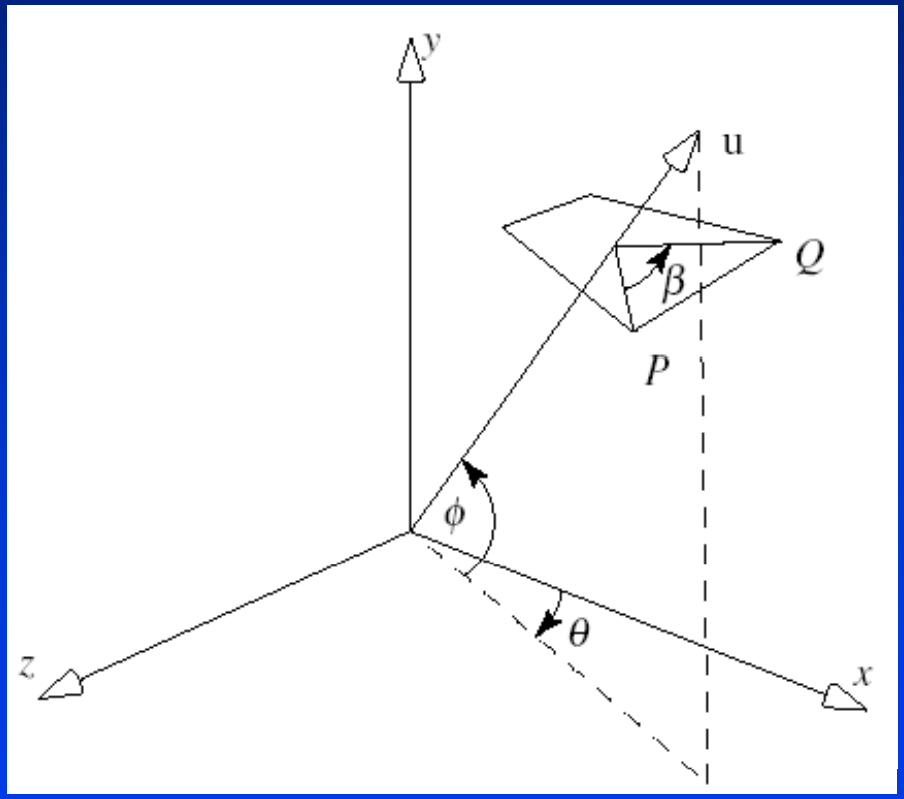
Axis: **u**

Point: P

Angle: β

Method:

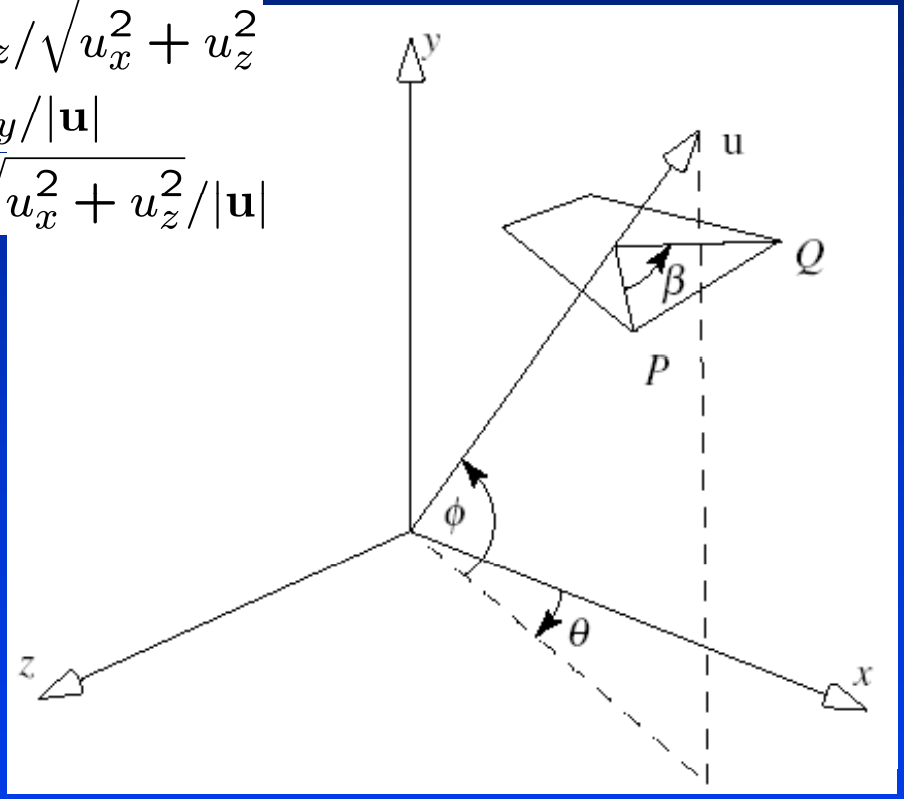
1. *Two rotations to align* **u** *with x-axis*
2. *Do x-roll by β*
3. *Undo the alignment*



Derivation

1. $R_z(-\phi)R_y(\theta)$
2. $R_x(\beta)$
3. $R_y(-\theta)R_z(\phi)$

$$cos(\theta) = u_x/\sqrt{u_x^2 + u_z^2}$$
$$sin(\theta) = u_z/\sqrt{u_x^2 + u_z^2}$$
$$sin(\phi) = u_y/|\mathbf{u}|$$
$$cos(\phi) = \sqrt{u_x^2 + u_z^2}/|\mathbf{u}|$$



Altogether:

$R_y(-\theta)R_z(\phi)\ R_x(\beta)\ R_z(-\phi)R_y(\theta)$

We can add translation too if the axis is not through the origin

Properties of affine transformations

1. *Preservation of affine combinations of points.*
2. *Preservation of lines and planes.*
3. *Preservation of parallelism of lines and planes.*
4. *Relative ratios are preserved*
5. *Affine transformations are composed of elementary ones.*

General form

Rotation, Scaling, Shear

Translation

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

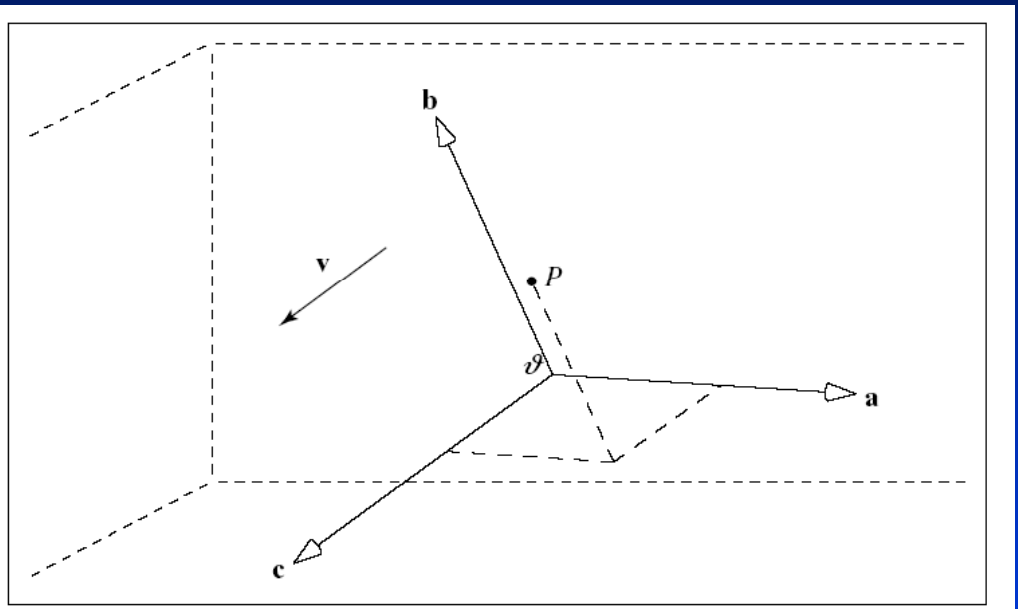
Transformations of Coordinate systems

Coordinate systems consist of vectors and an origin, therefore we can transform them just like points and vectors.

Alternative way to think of transformations

Reminder: Coordinate systems

Coordinate system: (**a**,**b**,**c**,θ)



$$\mathbf{v} = (v_1, v_2, v_3) \rightarrow \mathbf{v} = v_1\mathbf{a} + v_2\mathbf{b} + v_3\mathbf{c}$$

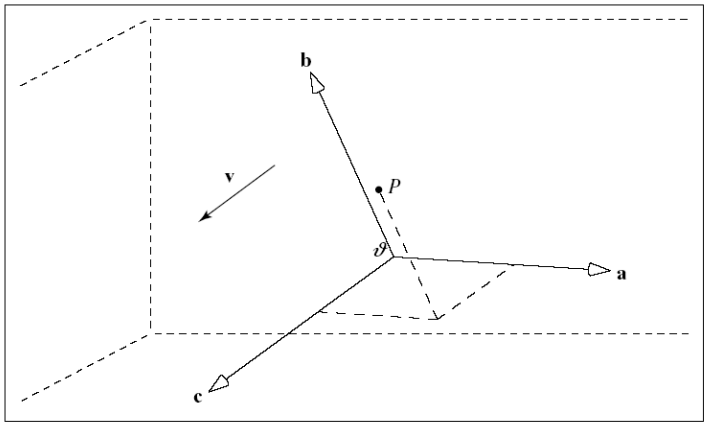
$$P = (p_1, p_2, p_3) \rightarrow P - \theta = p_1\mathbf{a} + p_2\mathbf{b} + p_3\mathbf{c}$$

$$P = \theta + p_1\mathbf{a} + p_2\mathbf{b} + p_3\mathbf{c}$$

Reminder: The homogeneous representation of points and vectors

$$\mathbf{v} = v_1\mathbf{a} + v_2\mathbf{b} + v_3\mathbf{c} \rightarrow \mathbf{v} = (\mathbf{a}, \mathbf{b}, \mathbf{c}, \theta) \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{pmatrix}$$

$$P = \theta + p_1\mathbf{a} + p_2\mathbf{b} + p_3\mathbf{c} \rightarrow P = (\mathbf{a}, \mathbf{b}, \mathbf{c}, \theta) \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{pmatrix}$$

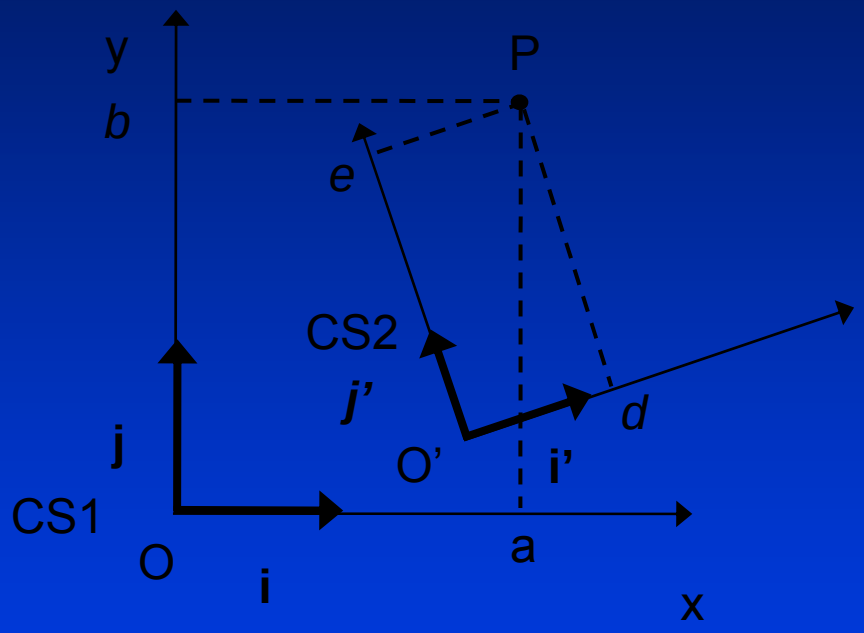


Transforming CS1 into CS2

What is the relationship between P in CS2 and P in CS1 if CS2 = T(CS1)?

$$CS1 : P = (a, b, c, 1)^T$$
$$CS2 : P = (d, e, f, 1)^T$$

$$O' = T(O),$$
$$\mathbf{i}' = T(\mathbf{i}),$$
$$\mathbf{j}' = T(\mathbf{j}),$$
$$\mathbf{k}' = T(\mathbf{k})$$



Derivation

By definition P is the linear combination of vectors $\mathbf{i}', \mathbf{j}', \mathbf{k}'$ and point O'.

$$P = d\mathbf{i}' + e\mathbf{j}' + f\mathbf{k}' + O'$$

In system CS1:

$$P_{CS1} = d\mathbf{i}'_{CS1} + e\mathbf{j}'_{CS1} + f\mathbf{k}'_{CS1} + O'_{CS1}$$

Derivation

$$P_{CS1} = d\mathbf{i}'_{CS1} + e\mathbf{j}'_{CS1} + f\mathbf{k}'_{CS1} + O'_{CS1}$$

We know that $(\mathbf{i}'_{CS1}, \mathbf{j}'_{CS1}, \mathbf{k}'_{CS1}, O'_{CS1}) = T((\mathbf{i}, \mathbf{j}, \mathbf{k}, O))$

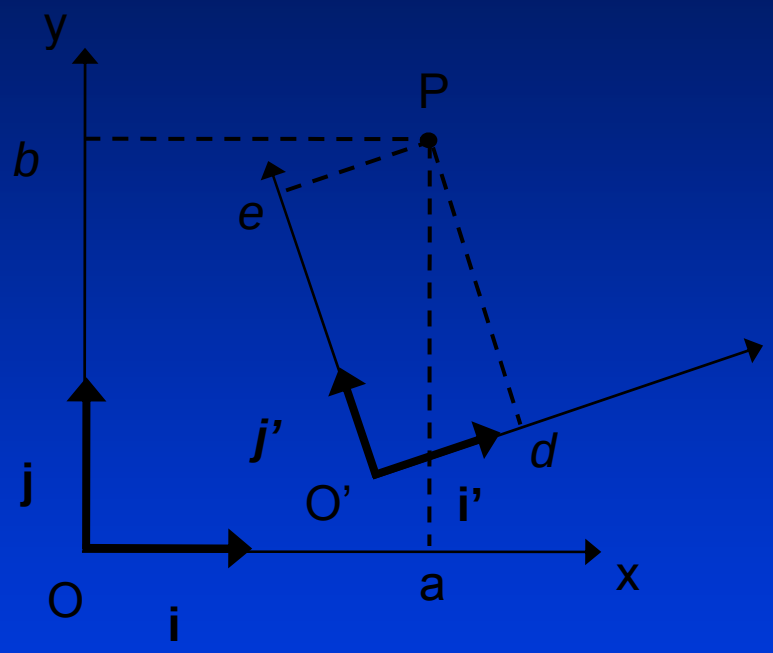
$$\begin{aligned}
P_{CS1} &= dT(\mathbf{i}) + eT(\mathbf{j}) + fT(\mathbf{k}) + T(O) \\
&= d(\mathbf{M}\mathbf{i}) + e(\mathbf{M}\mathbf{j}) + f(\mathbf{M}\mathbf{k}) + \mathbf{M}O \\
&= d(\mathbf{M}\begin{bmatrix}1\\0\\0\\0\end{bmatrix}) + e(\mathbf{M}\begin{bmatrix}0\\1\\0\\0\end{bmatrix}) + f(\mathbf{M}\begin{bmatrix}0\\0\\1\\0\end{bmatrix}) + \mathbf{M}\begin{bmatrix}0\\0\\0\\1\end{bmatrix} \\
&= \mathbf{M}\begin{bmatrix}d\\0\\0\\0\end{bmatrix} + \mathbf{M}\begin{bmatrix}0\\e\\0\\0\end{bmatrix} + \mathbf{M}\begin{bmatrix}0\\0\\f\\0\end{bmatrix} + \mathbf{M}\begin{bmatrix}0\\0\\0\\1\end{bmatrix} \\
&= \mathbf{M}(\begin{bmatrix}d\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\e\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\f\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\1\end{bmatrix}) = \mathbf{M}\begin{bmatrix}d\\e\\f\\1\end{bmatrix}
\end{aligned}$$

P in CS1 vs P in CS2

Proof in pages 245,246 of [Hill]

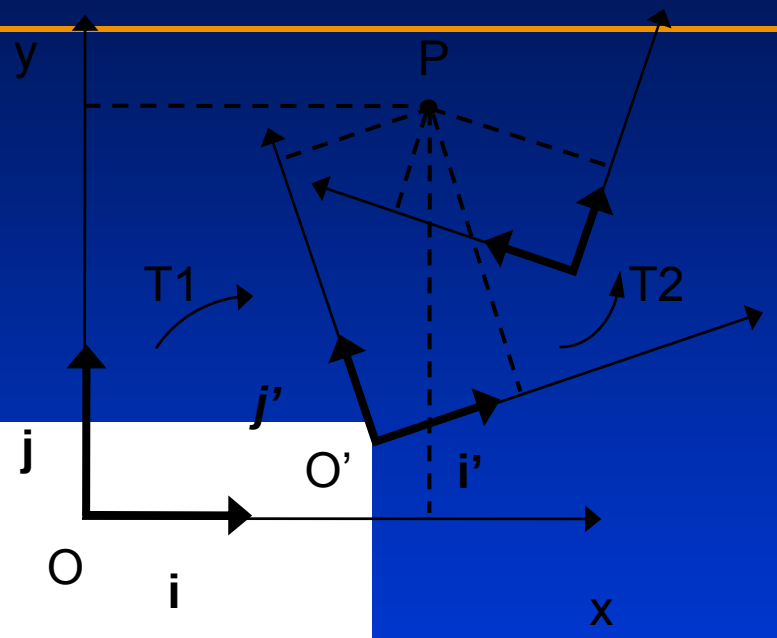
$$P_{CS1} = \mathbf{M} P_{CS2}$$

$$\begin{pmatrix} a \\ b \\ c \\ 1 \end{pmatrix} = \mathbf{M} \begin{pmatrix} d \\ e \\ f \\ 1 \end{pmatrix}$$



Successive transformations of CS

CS1 → CS2 → CS3

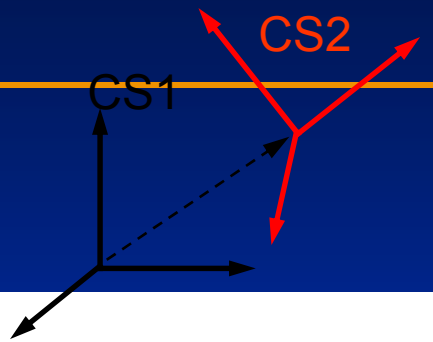


Working backwards:

$$P_{CS2} = M_2 P_{CS3} \rightarrow \begin{pmatrix} d \\ e \\ f \\ 1 \end{pmatrix} = M_2 \begin{pmatrix} g \\ h \\ m \\ 1 \end{pmatrix}$$

$$P_{CS1} = M_1 P_{CS2} \rightarrow \begin{pmatrix} a \\ b \\ c \\ 1 \end{pmatrix} = M_1 \begin{pmatrix} d \\ e \\ f \\ 1 \end{pmatrix} = M_1 M_2 \begin{pmatrix} g \\ h \\ m \\ 1 \end{pmatrix}$$

Transformations as a change of basis



We know the basis vectors and we know that

$$P_{CS1} = MP_{CS2}$$

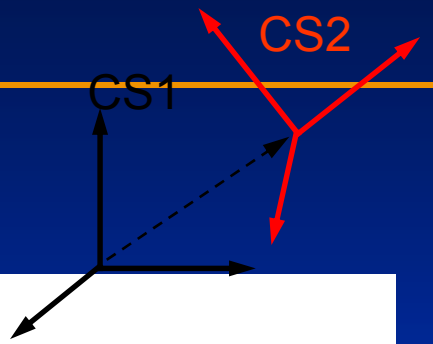
What is M with respect to the basis vectors?

$$P_{CS2} = a\mathbf{i}'_{CS2} + b\mathbf{j}'_{CS2} + c\mathbf{k}'_{CS2} + O'_{CS2} = a\begin{bmatrix}1\\0\\0\end{bmatrix} + b\begin{bmatrix}0\\1\\0\end{bmatrix} + c\begin{bmatrix}0\\0\\1\end{bmatrix} + \begin{bmatrix}0\\0\\0\end{bmatrix}$$

$$P_{CS1} = a\mathbf{i}'_{CS1} + b\mathbf{j}'_{CS1} + c\mathbf{k}'_{CS1} + O'_{CS1} = a\begin{bmatrix}i'_x\\i'_y\\i'_z\end{bmatrix} + b\begin{bmatrix}j'_x\\j'_y\\j'_z\end{bmatrix} + c\begin{bmatrix}k'_x\\k'_y\\k'_z\end{bmatrix} + \begin{bmatrix}O'_x\\O'_y\\O'_z\end{bmatrix}$$

$$P_{CS1} = \begin{bmatrix}x\\y\\z\\1\end{bmatrix} = \begin{bmatrix}i'_x & j'_x & k'_x & O'_x\\i'_y & j'_y & k'_y & O'_y\\i'_z & j'_z & k'_z & O'_z\\0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}a\\b\\c\\1\end{bmatrix} = MP_{CS2}$$

Transformations as a change of basis



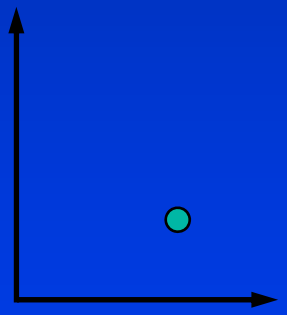
$$P_{CS1} = MP_{CS2}$$

$$P_{CS1} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & j'_x & k'_x & O'_x \\ i'_y & j'_y & k'_y & O'_y \\ i'_z & j'_z & k'_z & O'_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix} = MP_{CS2}$$

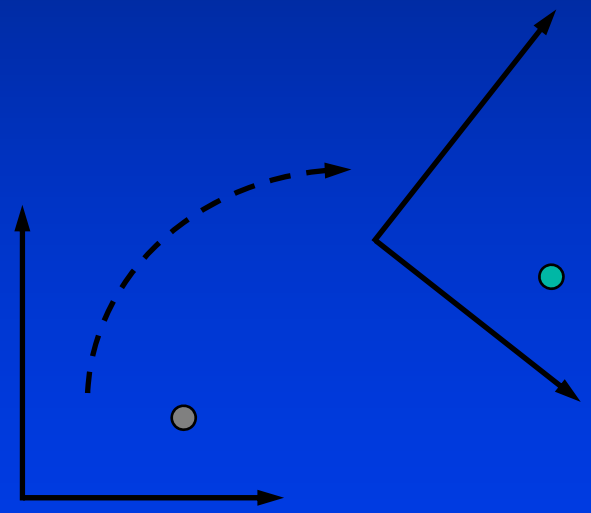
That is:

We can view transformations as a change of coordinate system

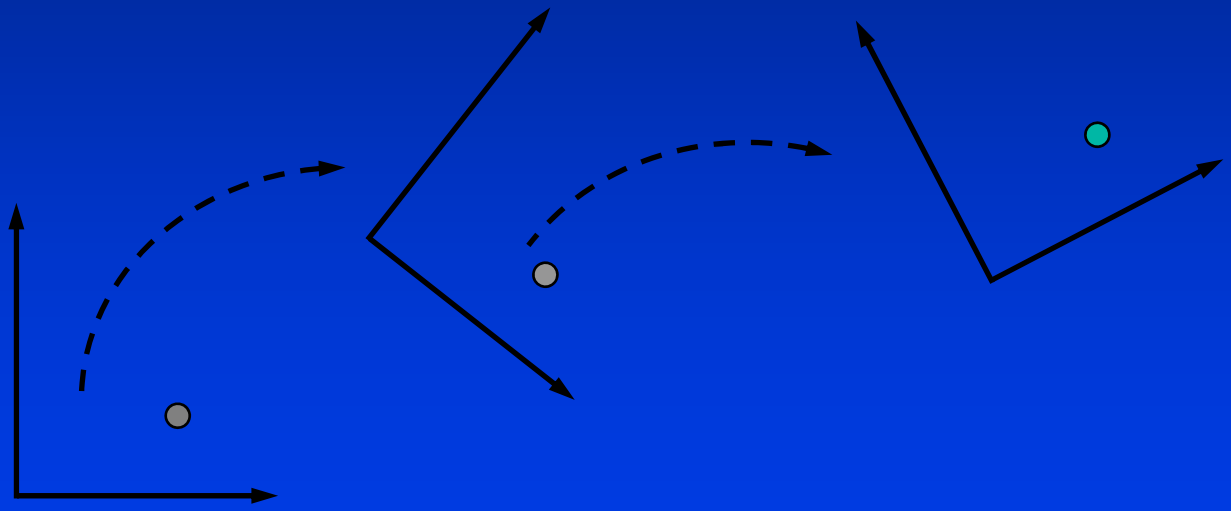
Transforming a point through transforming coordinate systems



Transforming a point through transforming coordinate systems



Transforming a point through transforming coordinate systems



Rule of thumb

Transforming a point P:

Transformations: T1,T2,T3

Matrix: M = M3 x M2 x M1

Point transformed by: MP

Succesive transformations happen with respect to the same CS

Transforming a CS

Transformations: T1, T2, T3

Matrix: M = M1 x M2 x M3

A point has original coordinates MP

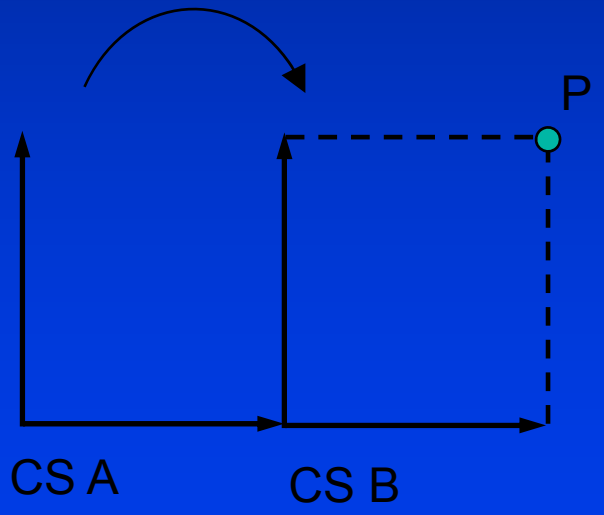
Each transformations happens with respect to the new CS.

Rule of thumb

To find the transformation matrix that transforms P from CSA coordinates to CSB coordinates, we find the sequence of transformations that align CSB to CSA accumulating matrices from left to right.

Explanation of this rule

Transformation M: ${}_{A}M_{B}$



If we think transforming systems, M takes CS A from the left and produces B on the right.

$\overrightarrow{{}_{A}M_{B}}$

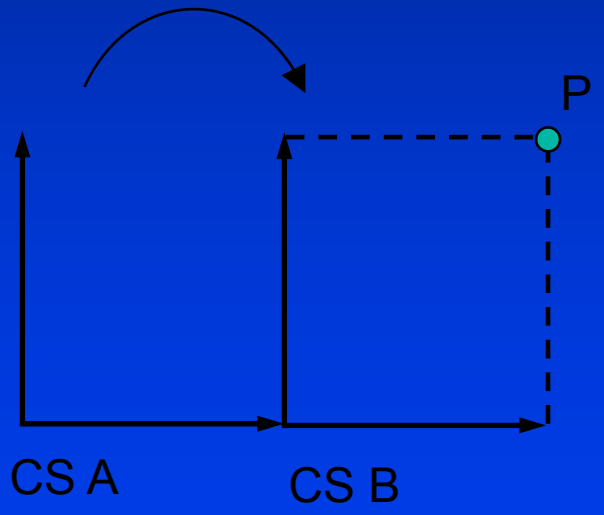
After this transformation we talk in B coordinates (right side).

If we think about points then we move the other way. M takes B on the right and produces the A coordinates on the left:

$\overleftarrow{{}_{A}M_{B}}$

Explanation of this rule

Transformation M: ${}_{A}M_{B}$

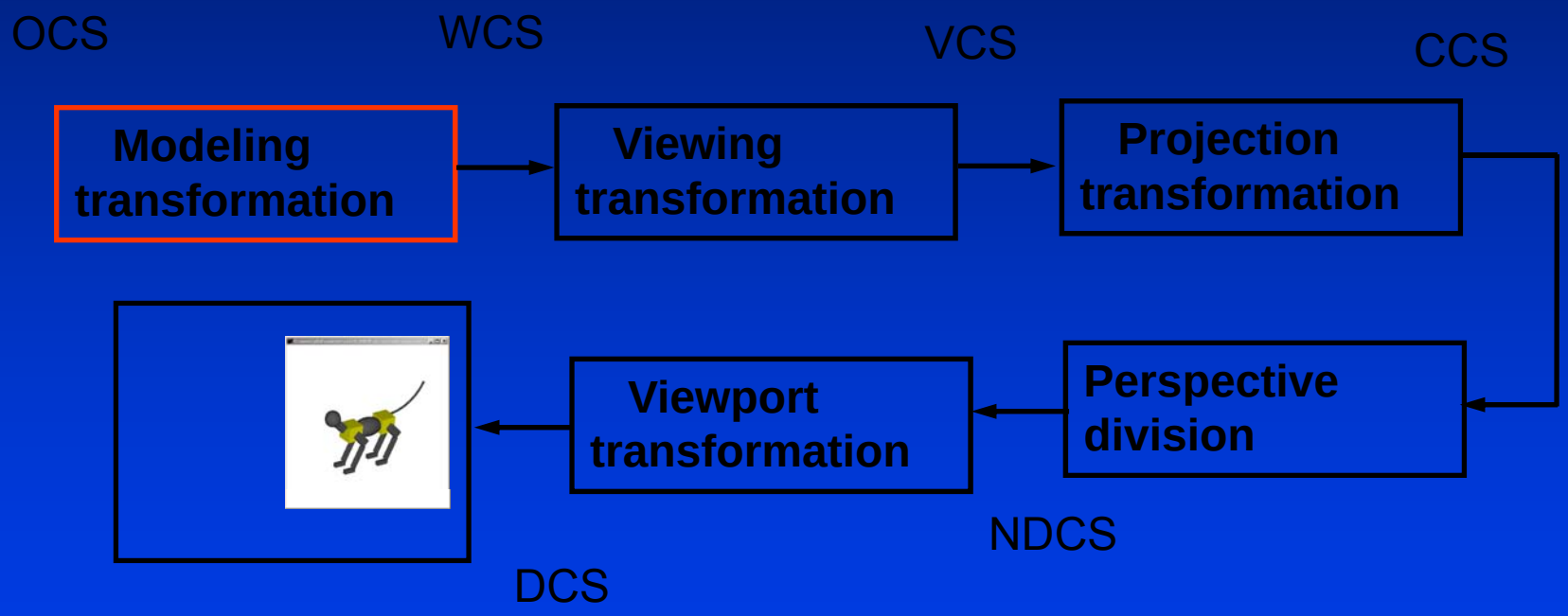


Take this simple example where to produce B we translate A by 1 on x axis.

$P_B = (1,1)$ $P_A = (2,1)$

If we move A by +1 to transform it into B then the coordinates of P with respect to the new system are shortened by 1 (B is closer to P than A by 1). So if we want to transform the coordinates of P from B to A we need to add 1 in x. Exactly what we need to do to transform system A to B.

Graphics Pipeline



Translation in OpenGL

glTranslate3f(GLfloat x, GLfloat y, GLfloat z) ;

glTranslate3d(GLdouble x, GLdouble y, GLdouble z);

$$\begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scaling in OpenGL

glScalef(GLfloat sx, GLfloat sy, GLfloat sz) ;

glScaled(GLdouble sx, GLdouble sy, GLdouble sz) ;

$$\begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation in OpenGL

glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z) ;

glRotated(GLdouble angle, GLdouble ux, GLdouble uy, GLdouble uz) ;

(Matrix in the next slide)

Composition of transformations in OpenGL

Successively transforming the coordinate system

$M = M_1 M_2 M_3 \ldots M_n$

$P_{world} = M_{Pobj}$

OpengGL Modelview Matrix

Each transformation post multiplies the current modelview matrix CM

```
glMatrixMode(GL_MODELVIEW) ;
glLoadIdentity() ;              // CM = I
glRotatef(45, 0,0,1) ;          // CM = I*Rz(45) ;
glTranslatef(1,1,1) ;           // CM = CM*T(1,1,1) =
                                //    = I*Rz(45) *T(1,1,1)
glScale(2,1,1) ;                // CM = CM *S(2,1,1) = I*Rz*T*S
```

Arbitrary matrices

Arbitrary affine (or not) transformations

glLoadMatrixf(GLfloat *M) ; // CM = M

glLoadMatrixd(GLdouble *M) ; // CM = M

glMultMatrixf(GLfloat *M) ; // CM = CM*M

glMultMatrixd(GLfloat *M) ; // CM = CM*M

Tricky Point

There are no multi-dimensional arrays in c.

Column-major order vs. row-major order.

OpenGL uses column major order that is:

$$\mathbf{float}\quad m[16] = a0, a1, a2, a3 \ldots, a15;$$

$$\mathbf{becomes}:$$

$$\begin{bmatrix} a0 & a4 & a8 & a12 \\ a1 & a5 & a9 & a13 \\ a2 & a6 & a10 & a14 \\ a3 & a7 & a11 & a15 \end{bmatrix}$$

Feedback

GLdouble m[16] ; glGetDoublev(GL_MODELVIEW_MATRIX,m) ;

GLfloat m[16] ; glGetFloatv(GL_MODELVIEW_MATRIX,m) ;

Matrix Stack

Why a stack?

- Reuse of transformations
- Control the effect of transformations
- Hierarchical structures

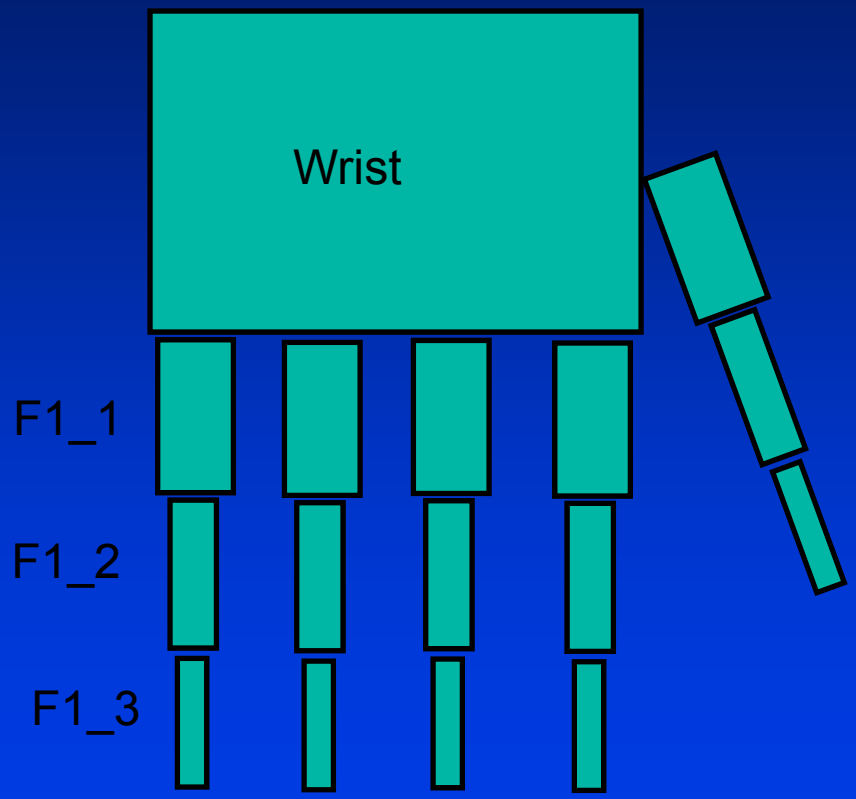
Manipulating the stack

- glPushMatrix() ;
- glPopMatrix() ;

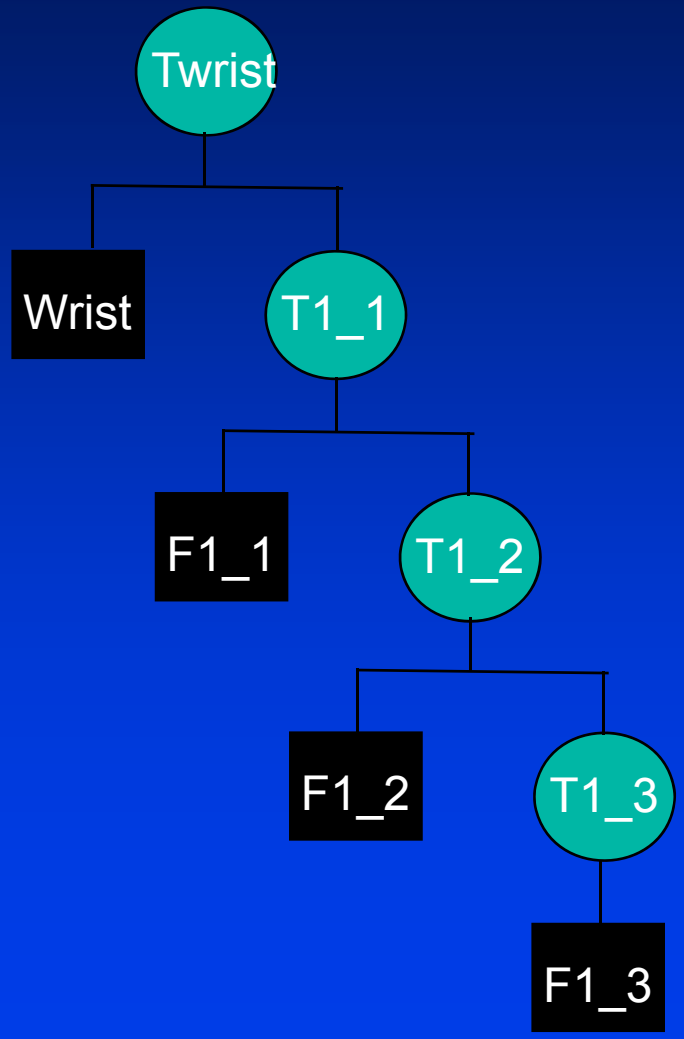
Example

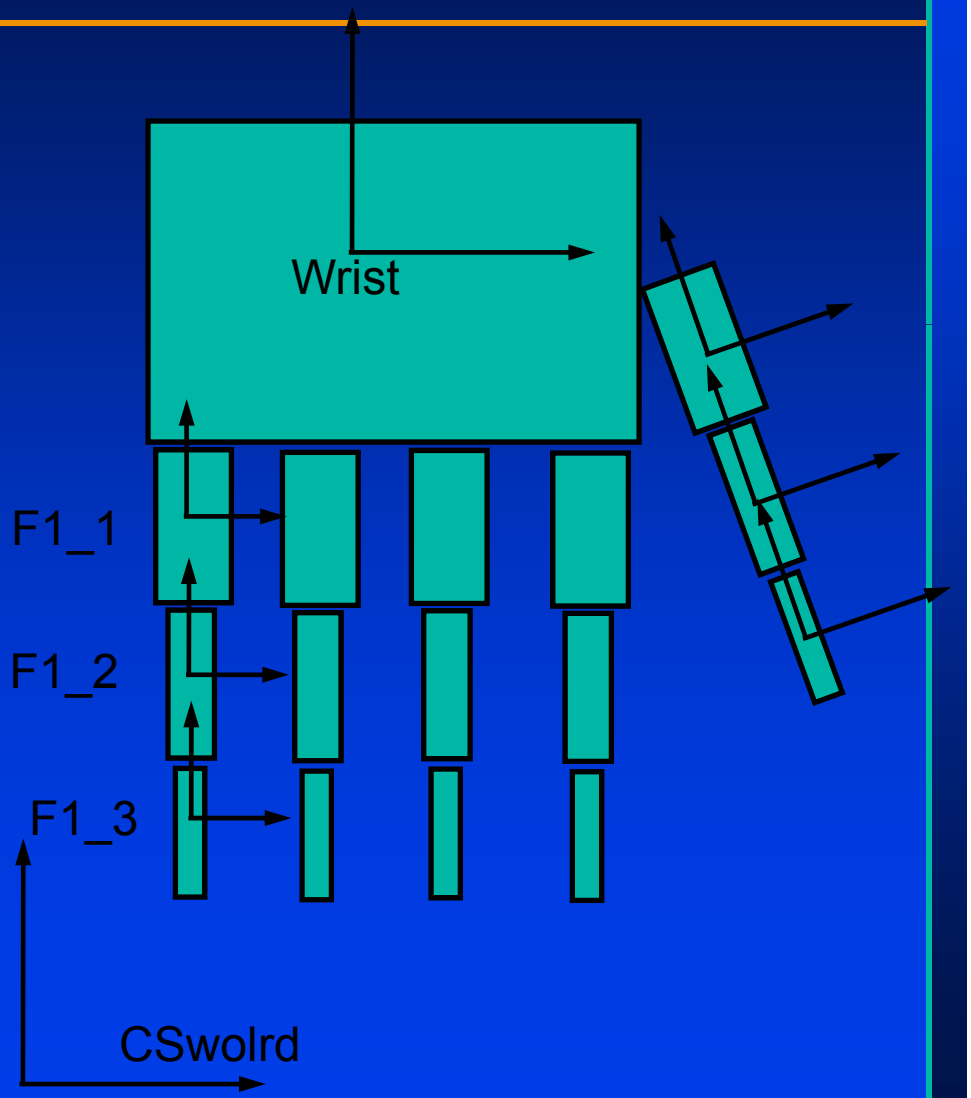
Wrist and 5 fingers

We want the fingers to stay attached to the wrist as the wrist moves.



Hierarchy


Twrist
Wrist
T1_1
F1_1
T1_2
F1_2
T1_3
F1_3

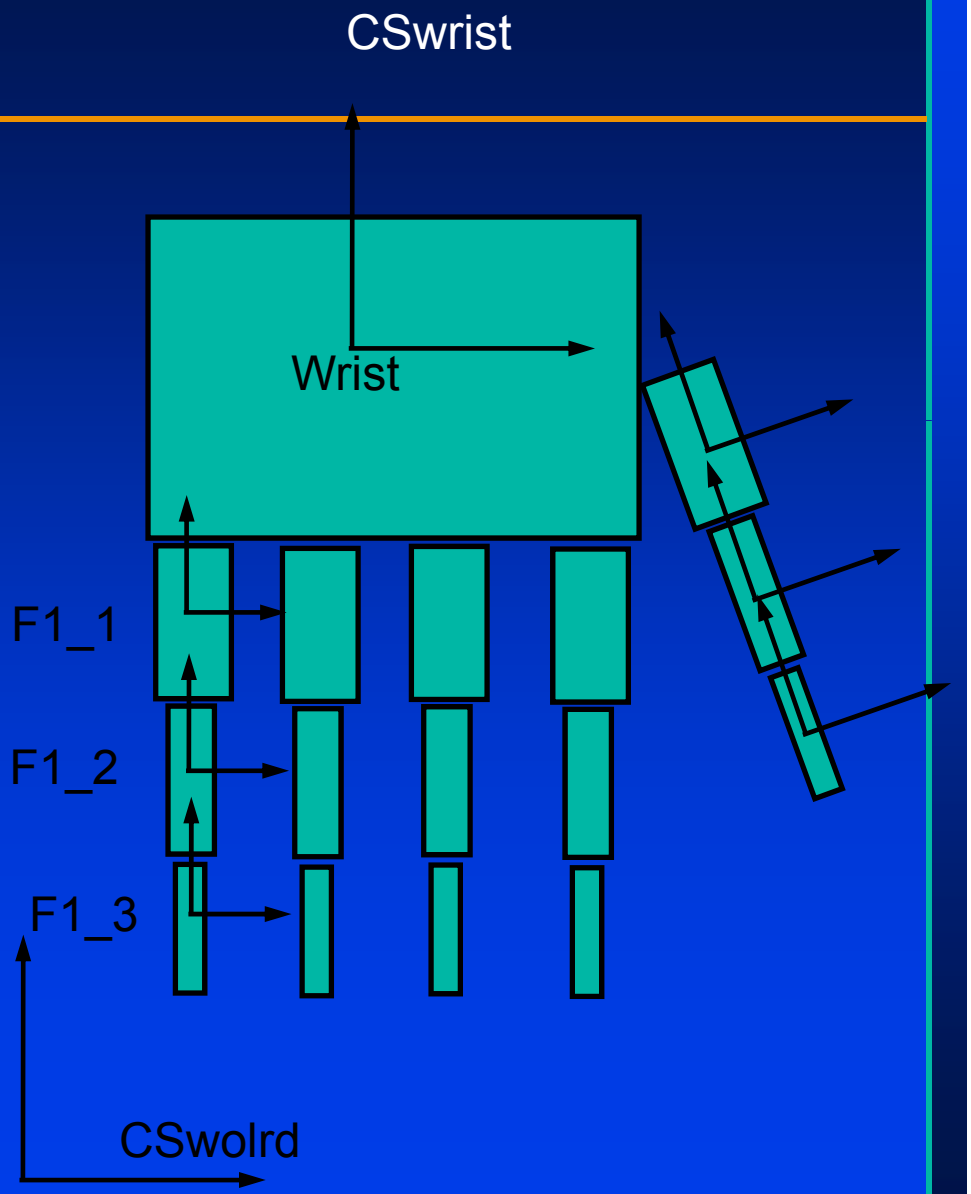

CSwrist
Wrist
F1_1
F1_2
F1_3
CSwolrd

Hierarchy

CSF1_1 = T1_1(CSwrist)

CSF1_2 = T1_2(CSF1_1)

CSF1_3 = T1_3(CSF1_2)



How to think about transformations

OpenGL code

- Transformations of coordinate systems TOP to BOTTOM
- Transformations of objects BOTTOM to TOP

Which one do we use to think of transformations?

- Whichever we like
- Usually both

Example:

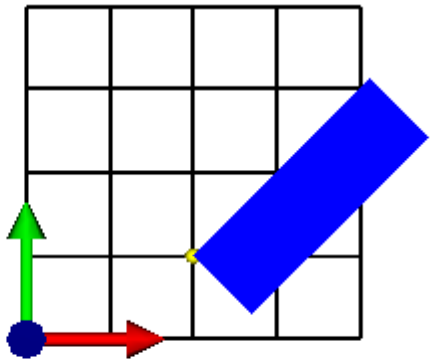
Given a unit cube center at the origin create a cube as shown in the next slide

c:\users\pfal\courses\cs174\code\transfExplanation\anim.exe

OpenGL Code:

```
glTranslatef(2,1,0);
glRotatef(45,0,0,1);
glScalef(3,1,1);
glTranslatef(0.5,-0.5,0);
drawCube() ;
```

Result:

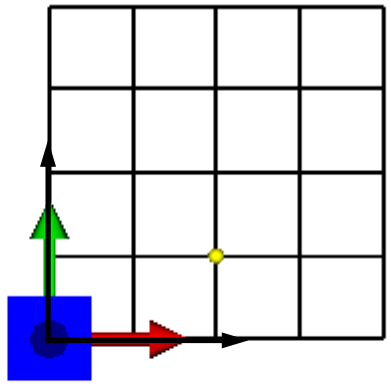


c:\users\pfal\courses\cs174\code\transfExplanation\anim.exe

OpenGL Code:

```
glTranslatef(2,1,0);
glRotatef(45,0,0,1);
glScalef(3,1,1);
glTranslatef(0.5,-0.5,0);
drawCube() ;
```

CS Method: TOP to BOTTOM thinking



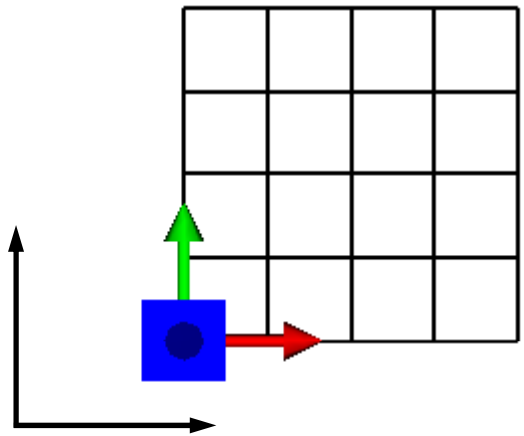
Each new transf with respect to the NEW system!

c:\users\pfal\courses\cs174\code\transfExplanation\anim.exe

OpenGL Code:

```
glTranslatef(2,1,0);
glRotatef(45,0,0,1);
glScalef(3,1,1);
glTranslatef(0.5,-0.5,0);
drawCube() ;
```

CS Method: TOP to BOTTOM thinking



glTrans(2,1,0)

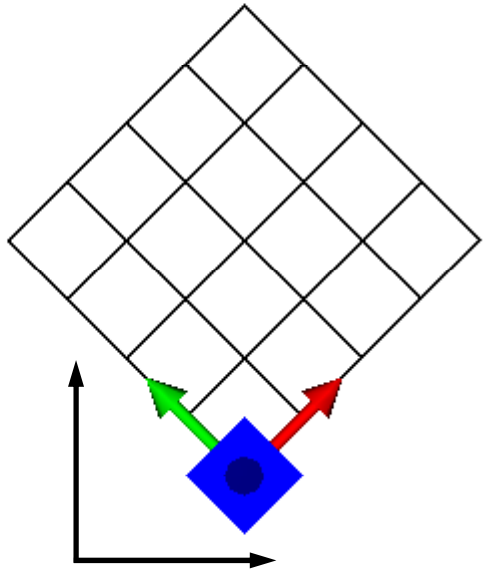
Each new transf with respect to the NEW system!

c:\users\pfal\courses\cs174\code\transfExplanation\anim.exe

OpenGL Code:

```
glTranslatef(2,1,0);
glRotatef(45,0,0,1);
glScalef(3,1,1);
glTranslatef(0.5,-0.5,0);
drawCube() ;
```

CS Method: TOP to BOTTOM thinking



glTrans(2,1,0)glRot(45,0,0,1)

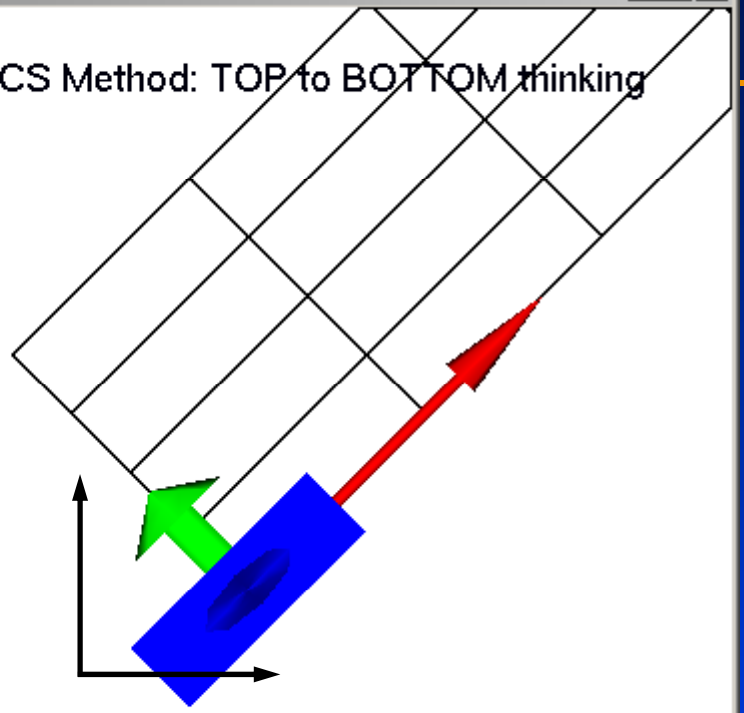
Each new transf with respect to the NEW system!

c:\users\pfal\courses\cs174\code\transfExplanation\anim.exe

OpenGL Code:

```
glTranslatef(2,1,0);
glRotatef(45,0,0,1);
glScalef(3,1,1);
glTranslatef(0.5,-0.5,0);
drawCube() ;
```

CS Method: TOP to BOTTOM thinking



glTrans(2,1,0)glRot(45,0,0,1)glScale(3,1,1)

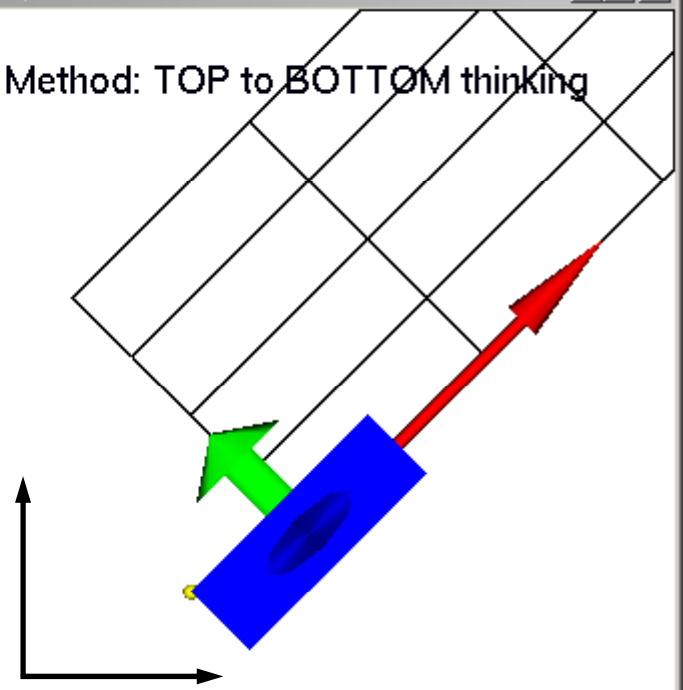
Each new transf with respect to the NEW system!

c:\users\pfal\courses\cs174\code\transfExplanation\anim.exe

OpenGL Code:

```
glTranslatef(2,1,0);
glRotatef(45,0,0,1);
glScalef(3,1,1);
glTranslatef(0.5,-0.5,0);
drawCube() ;
```

CS Method: TOP to BOTTOM thinking



glTrans(2,1,0)glRot(45,0,0,1)glScale(3,1,1)glTrans(0.5,-0.5,0)

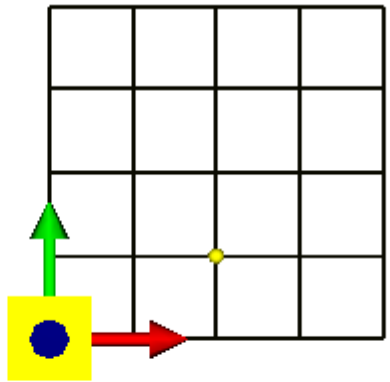
Each new transf with respect to the NEW system!

c:\users\pfal\courses\cs174\code\transfExplanation\anim.exe

OpenGL Code:

```
glTranslatef(2,1,0);
glRotatef(45,0,0,1);
glScalef(3,1,1);
glTranslatef(0.5,-0.5,0);
drawCube() ;
```

Object Method: Thinking BOTTOM to TOP



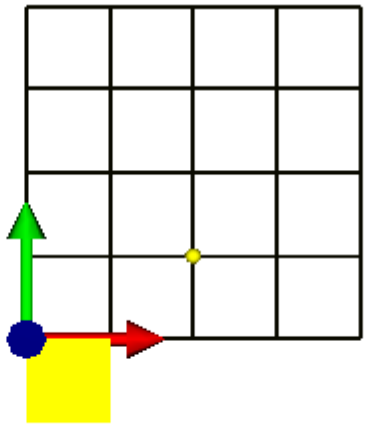
Each new transf with respect to the same system!

c:\users\pfal\courses\cs174\code\transfExplanation\anim.exe

OpenGL Code:

```
glTranslatef(2,1,0);
glRotatef(45,0,0,1);
glScalef(3,1,1);
glTranslatef(0.5,-0.5,0);
drawCube() ;
```

Object Method: Thinking BOTTOM to TOP



glTrans(0.5,-0.5,0)

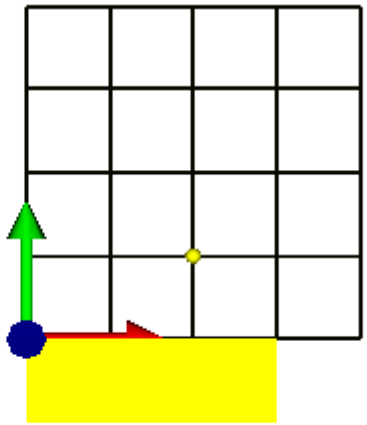
Each new transf with respect to the same system!

c:\users\pfal\courses\cs174\code\transfExplanation\anim.exe

OpenGL Code:

```
glTranslatef(2,1,0);
glRotatef(45,0,0,1);
glScalef(3,1,1);
glTranslatef(0.5,-0.5,0);
drawCube() ;
```

Object Method: Thinking BOTTOM to TOP



glScale(3,1,1)glTrans(0.5,-0.5,0)

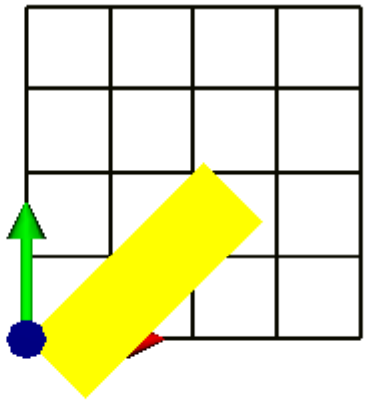
Each new transf with respect to the same system!

c:\users\pfal\courses\cs174\code\transfExplanation\anim.exe

OpenGL Code:

```
glTranslatef(2,1,0);
glRotatef(45,0,0,1);
glScalef(3,1,1);
glTranslatef(0.5,-0.5,0);
drawCube() ;
```

Object Method: Thinking BOTTOM to TOP



glRot(45,0,0,1)glScale(3,1,1)glTrans(0.5,-0.5,0)

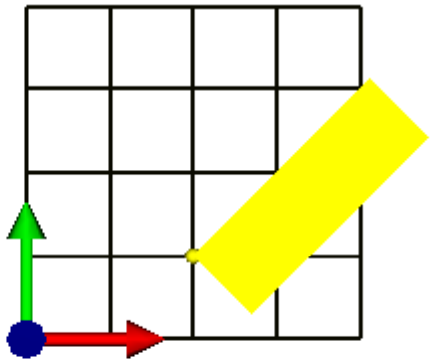
Each new transf with respect to the same system!

c:\users\pfal\courses\cs174\code\transfExplanation\anim.exe

OpenGL Code:

```
glTranslatef(2,1,0);
glRotatef(45,0,0,1);
glScalef(3,1,1);
glTranslatef(0.5,-0.5,0);
drawCube() ;
```

Object Method: Thinking BOTTOM to TOP



glTrans(2,1,0)glRot(45,0,0,1)glScale(3,1,1)glTrans(0.5,-0.5,0)

Each new transf with respect to the same system!

Hybrid way of thinking

Use TOP to BOTTOM to position a coordinate system

Then use BOTTOM to TOP to position the objects within that system

Often it is easier to do it in the opposite order