University of British Columbia

CPSC 314 Computer Graphics

Jan-Apr 2008

Tamara Munzner

Transformations III

Week 3, Mon Jan 21

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2008

Readings for Jan 16-25

- FCG Chap 6 Transformation Matrices
  - *except* 6.1.6, 6.3.1
- FCG Sect 13.3 Scene Graphs
- RB Chap Viewing
  - Viewing and Modeling Transforms *until* Viewing Transformations
  - Examples of Composing Several Transformations *through* Building an Articulated Robot Arm
- RB Appendix Homogeneous Coordinates and Transformation Matrices
  - *until* Perspective Projection
- RB Chap Display Lists

News

- Homework 1 out today

Review: 3D Transformations

shear(hxy,hxz,hyx,hyz,hzx,hzy)
$$\begin{bmatrix} 1 & hyx & hzx & 0 \\ hxy & 1 & hzy & 0 \\ hxz & hyz & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

translate(a,b,c)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & & & a \\ & 1 & & b \\ & & 1 & c \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

scale(a,b,c)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & & & \\ & b & & \\ & & c & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$\text{Rotate}(x, \theta)$

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

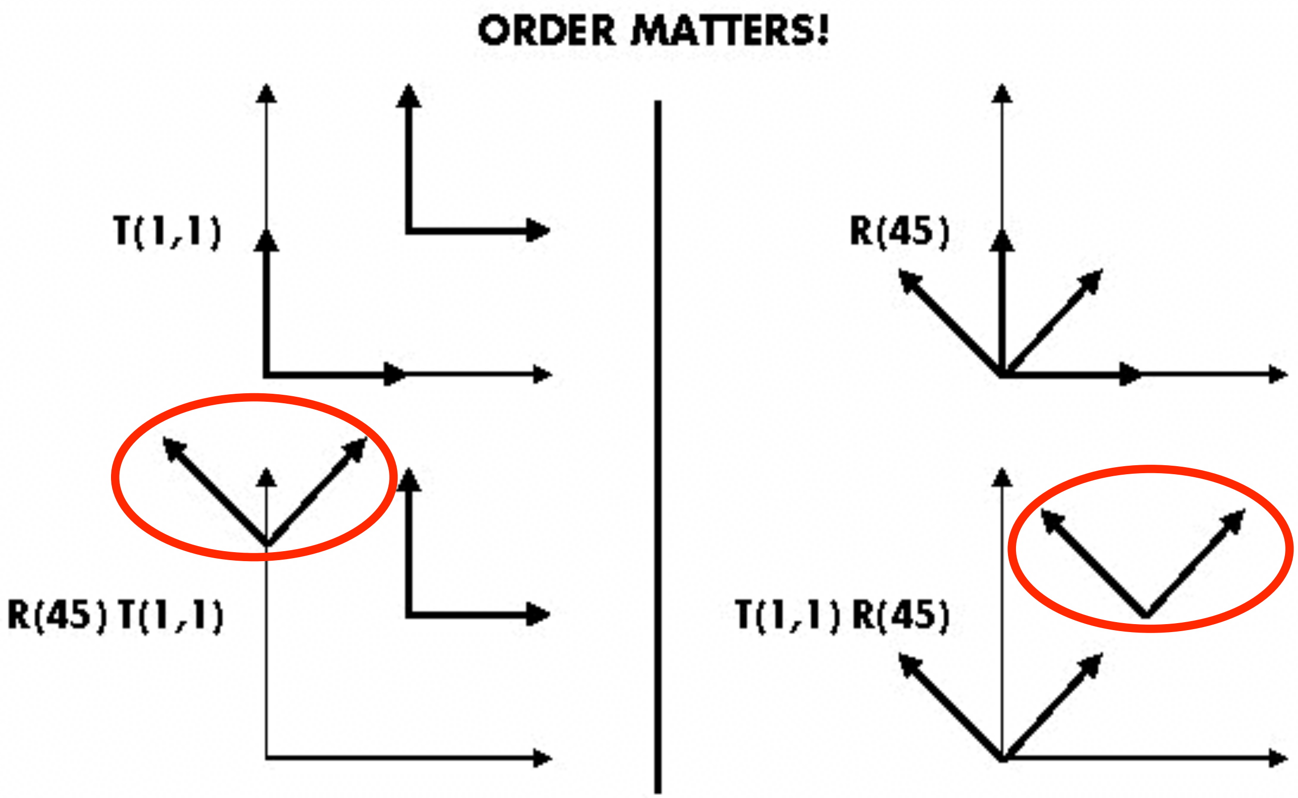
$\text{Rotate}(y, \theta)$

$$\begin{bmatrix} \cos\theta & & \sin\theta & \\ & 1 & & \\ -\sin\theta & & \cos\theta & \\ & & & 1 \end{bmatrix}$$

$\text{Rotate}(z, \theta)$

$$\begin{bmatrix} \cos\theta & -\sin\theta & & \\ \sin\theta & \cos\theta & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

Review: Composing Transformations



Ta Rb != Rb Ta

Review: Composing Transformations

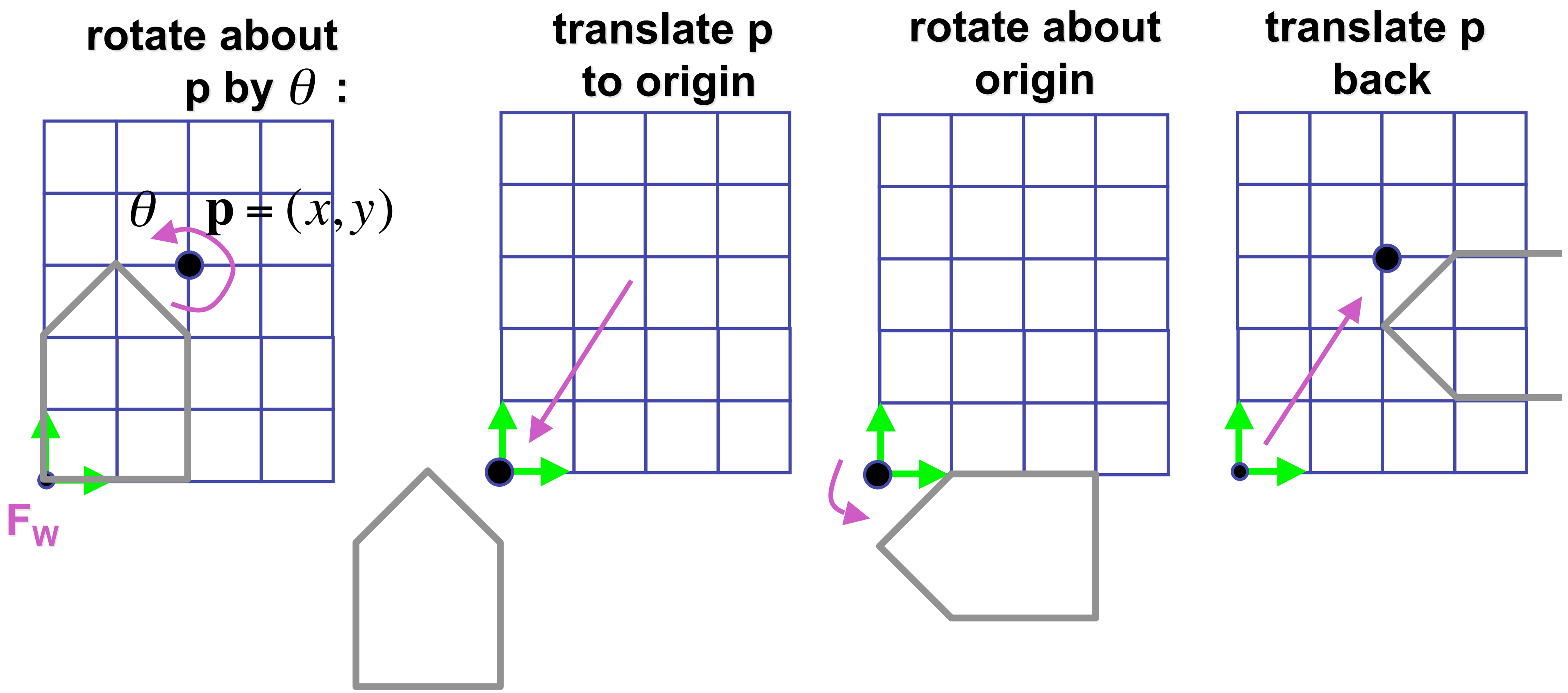
$$\mathbf{p}' = \mathbf{T}\mathbf{R}\mathbf{p}$$

- which direction to read?
  - right to left
    - interpret operations wrt fixed coordinates
    - moving object
  - left to right **OpenGL pipeline ordering!**
    - interpret operations wrt local coordinates
    - changing coordinate system
    - OpenGL updates current matrix with postmultiply
      - glTranslatef(2,3,0);
      - glRotatef(-90,0,0,1);
      - glVertexf(1,1,1);

Matrix Composition

- matrices are convenient, efficient way to represent series of transformations
  - general purpose representation
  - hardware matrix multiply
  - matrix multiplication is associative
    - **p′** = (T*(R*(S***p**)))
    - **p′** = (T*R*S)***p**
- procedure
  - correctly order your matrices!
  - multiply matrices together
  - result is one matrix, multiply vertices by this matrix
  - all vertices easily transformed with one matrix multiply

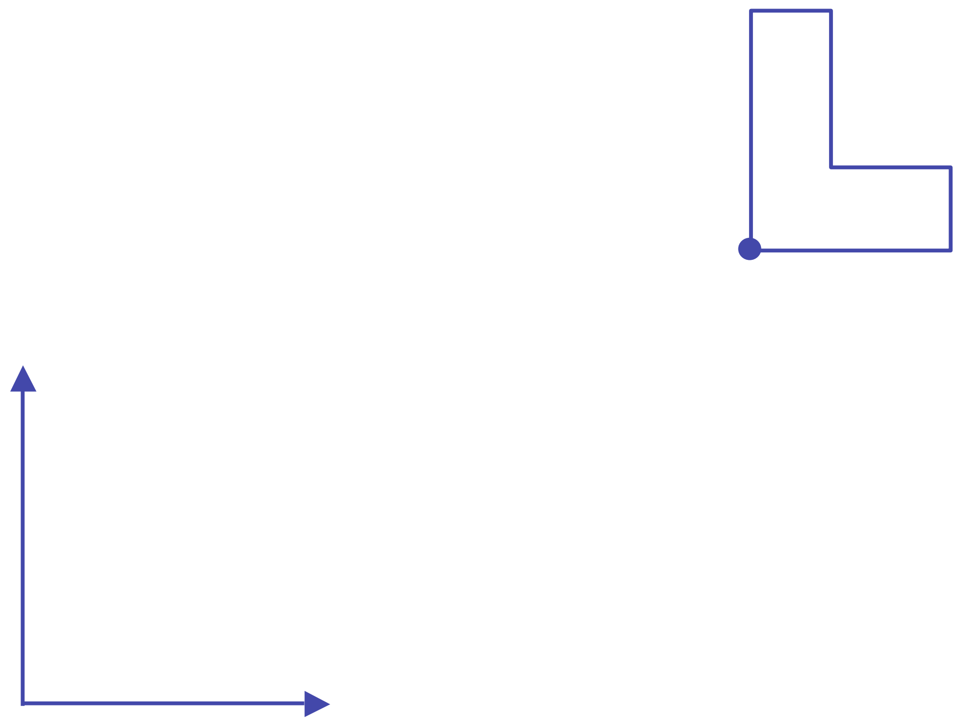
Rotation About a Point: Moving Object



$$\mathbf{T}(x, y, z)\,\mathbf{R}(z, \theta)\,\mathbf{T}(-x, -y, -z)$$

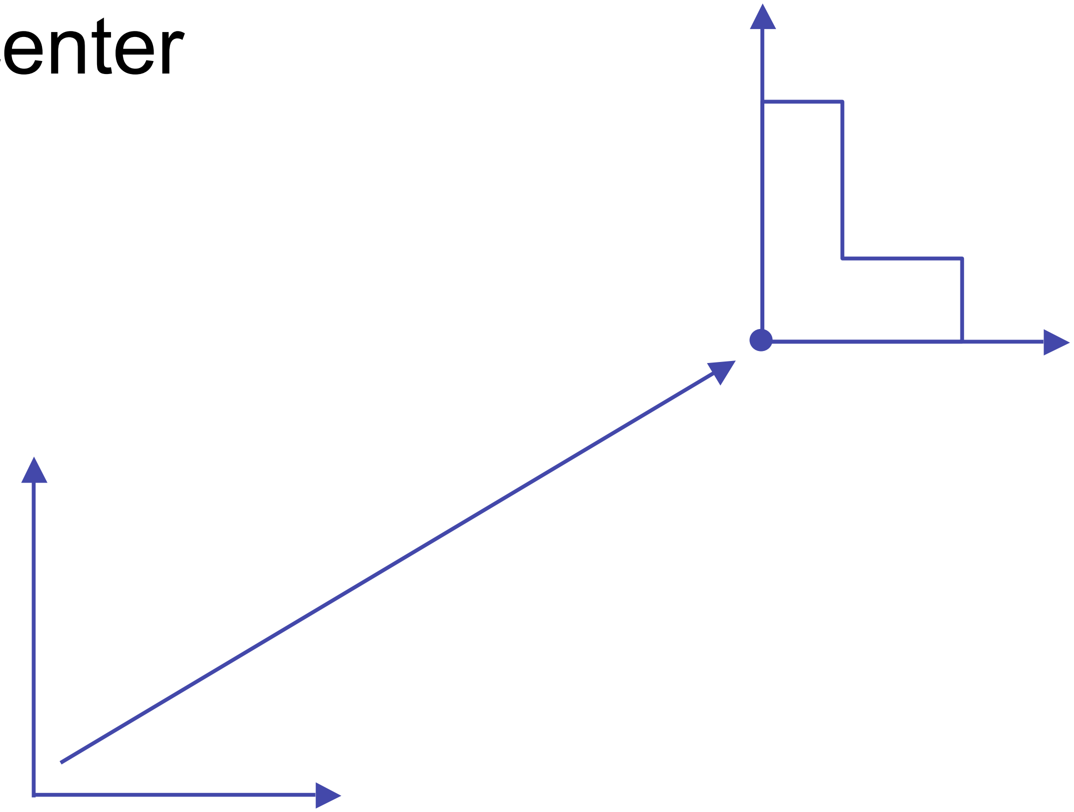
Rotation: Changing Coordinate Systems

- same example: rotation around arbitrary center



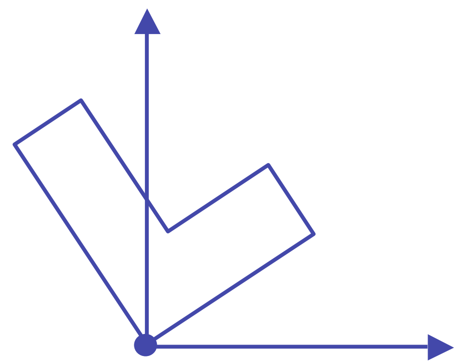
Rotation: Changing Coordinate Systems

- rotation around arbitrary center
  - step 1: translate coordinate system to rotation center



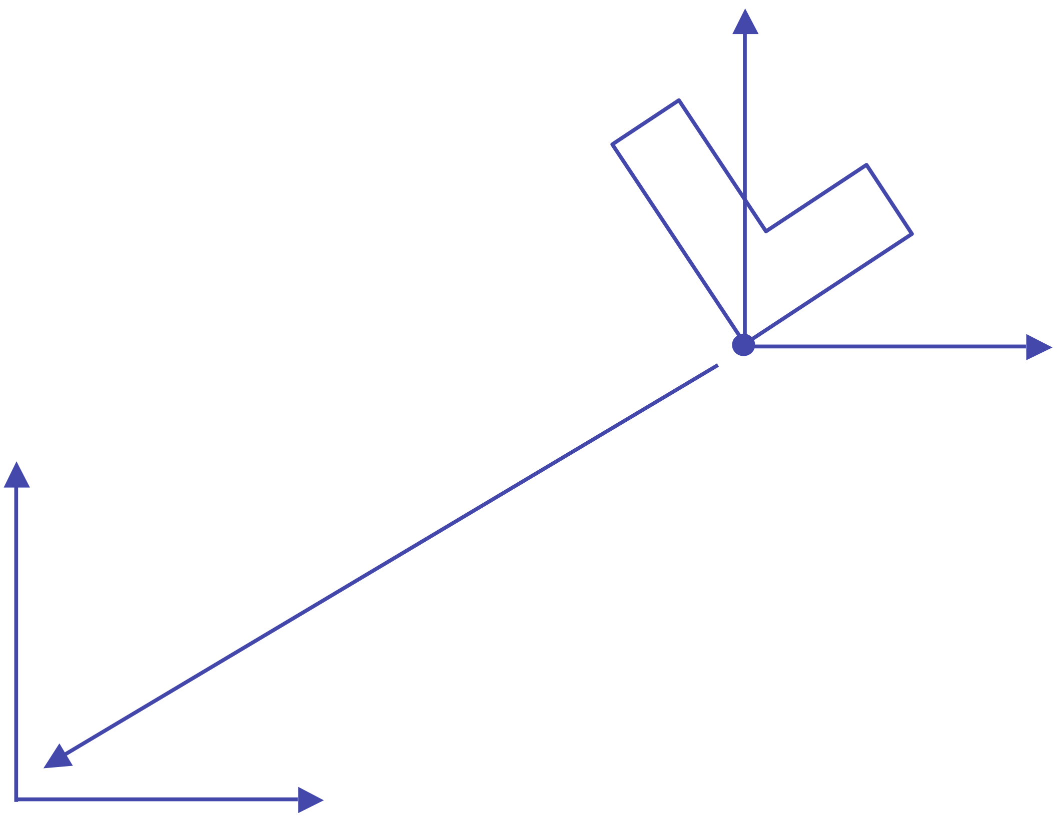
Rotation: Changing Coordinate Systems

- rotation around arbitrary center
  - step 2: perform rotation



Rotation: Changing Coordinate Systems

- rotation around arbitrary center
  - step 3: back to original coordinate system



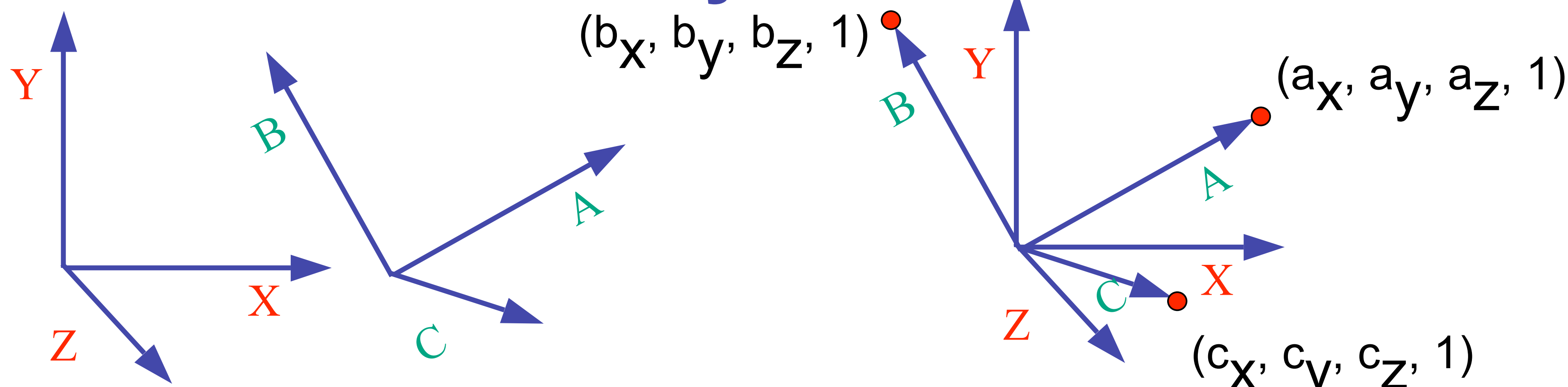
General Transform Composition

- transformation of geometry into coordinate system where operation becomes simpler
  - typically translate to origin
- perform operation
- transform geometry back to original coordinate system

Rotation About an Arbitrary Axis

- axis defined by two points
- translate point to the origin
- rotate to align axis with z-axis (or x or y)
- perform rotation
- undo aligning rotations
- undo translation

Arbitrary Rotation



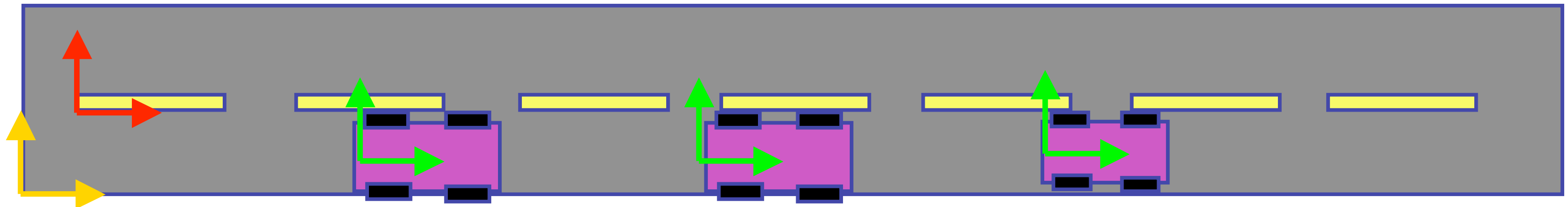
- arbitrary rotation: change of basis
  - given two orthonormal coordinate systems *XYZ* and *ABC*
    - *A*'s location in the XYZ coordinate system is $(a_x, a_y, a_z, 1)$, ...
- transformation from one to the other is matrix R whose columns are *A,B,C:*

$$R(X) = \begin{bmatrix} a_x & b_x & c_x & 0 \\ a_y & b_y & c_y & 0 \\ a_z & b_z & c_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = (a_x, a_y, a_z, 1) = A$$

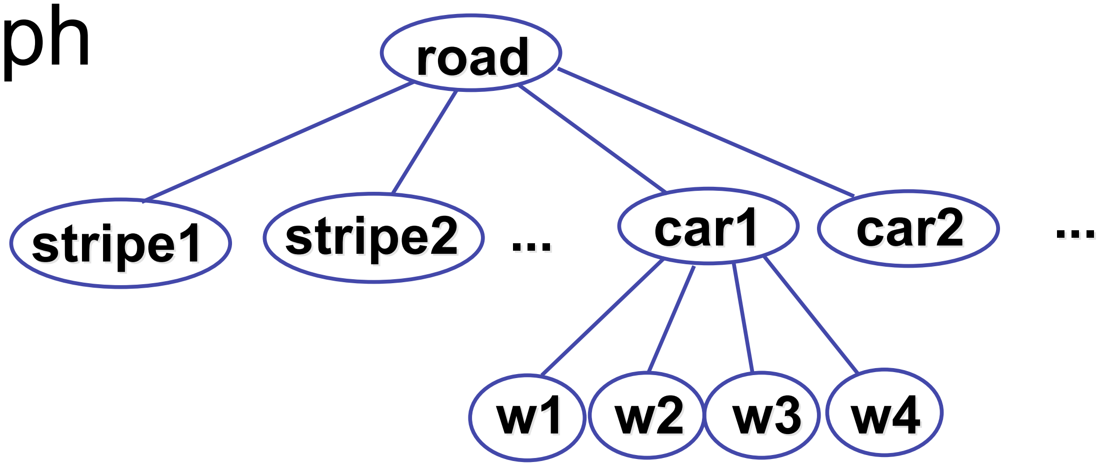
Transformation Hierarchies

Transformation Hierarchies

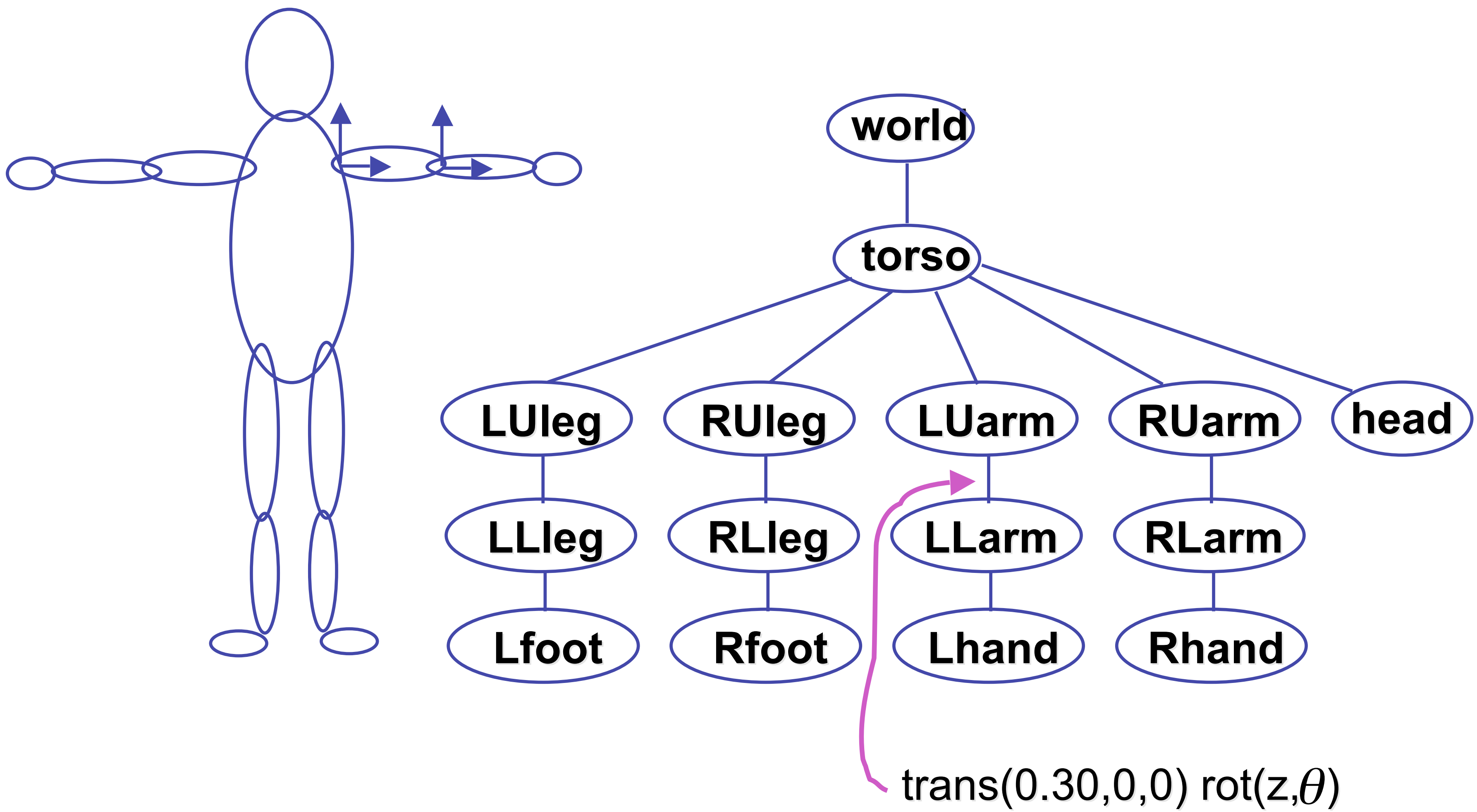
- scene may have a hierarchy of coordinate systems
  - stores matrix at each level with incremental transform from parent's coordinate system



- scene graph

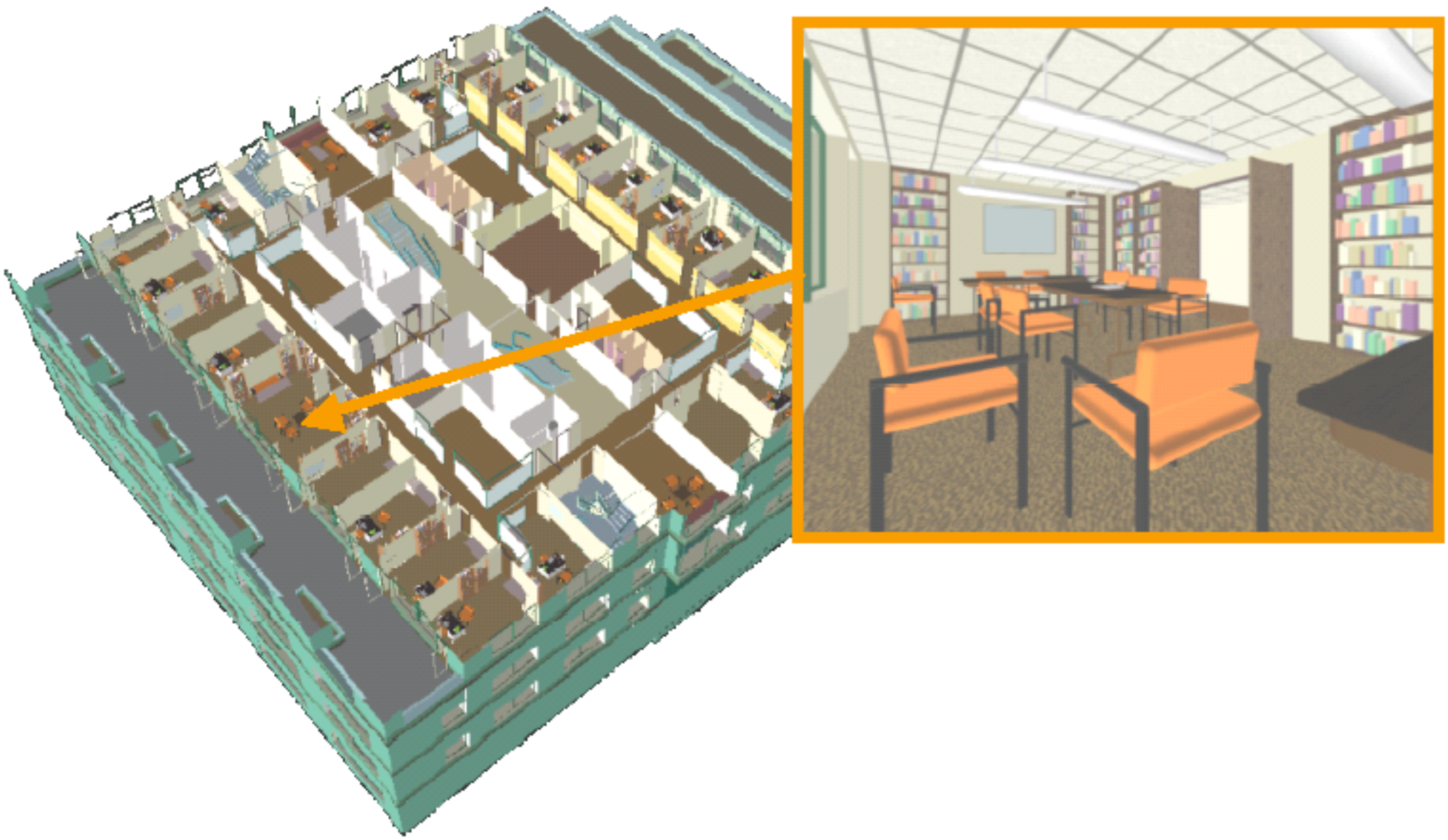


Transformation Hierarchy Example 1



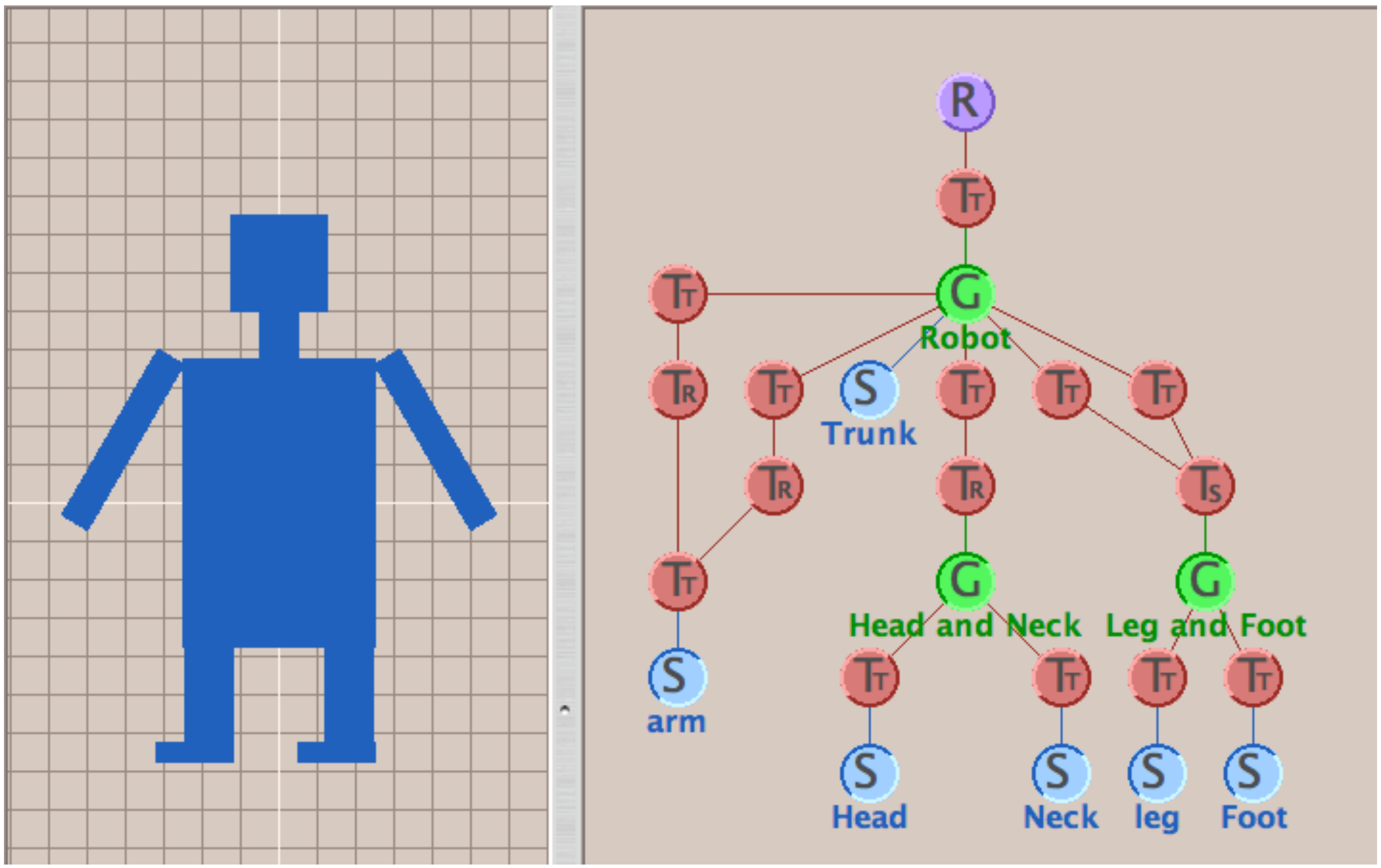
Transformation Hierarchy Example 2

- draw same 3D data with different transformations: instancing



Transformation Hierarchies Demo

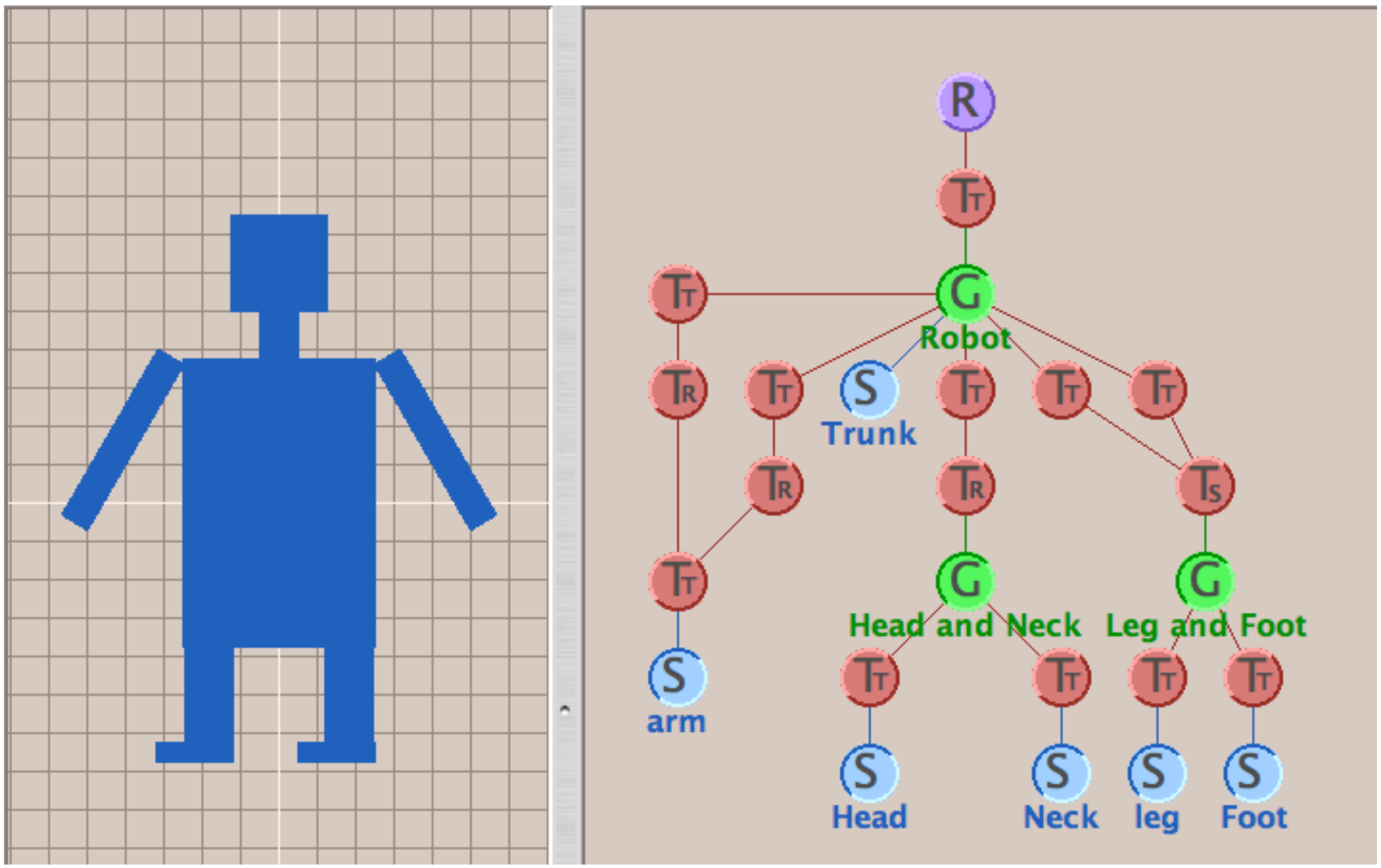
- transforms apply to graph nodes beneath



http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/scenegraphs.html

Transformation Hierarchies Demo

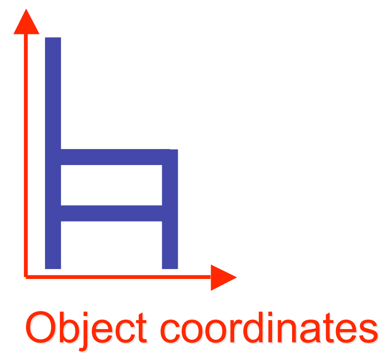
- transforms apply to graph nodes beneath

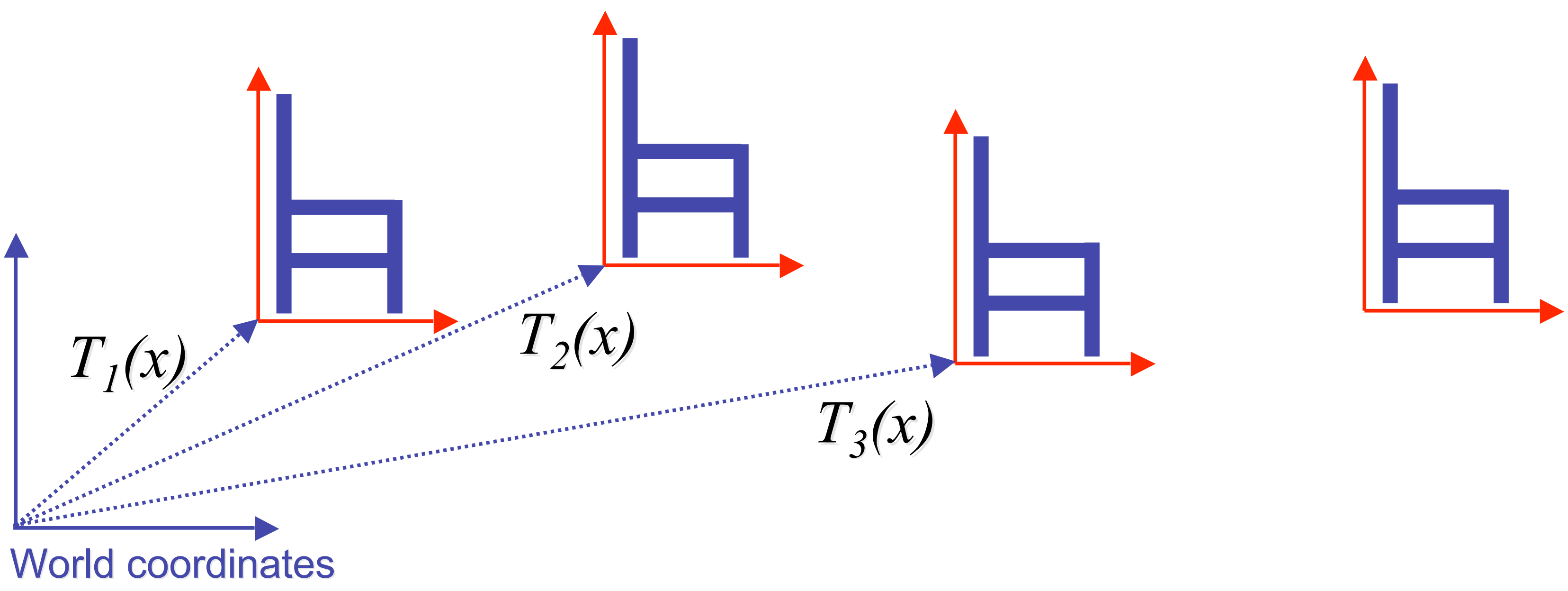


http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/scenegraphs.html

Matrix Stacks

- challenge of avoiding unnecessary computation
  - using inverse to return to origin
  - computing incremental T_1 -> T_2



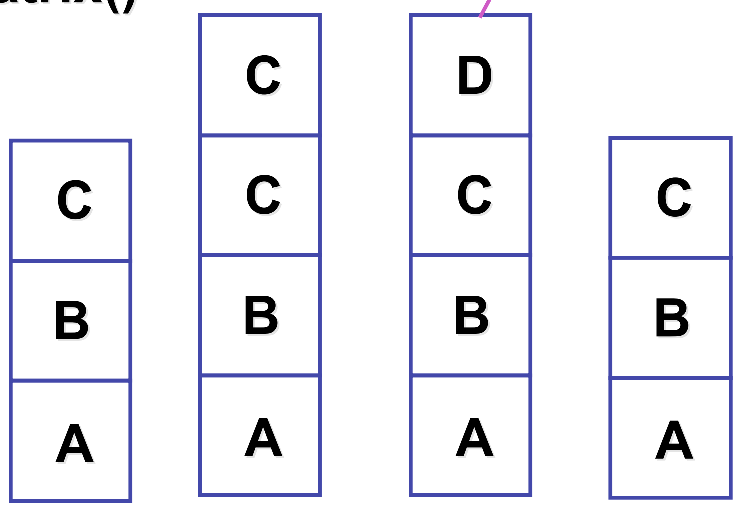


Matrix Stacks

glPushMatrix()

glPopMatrix()

D = C scale(2,2,2) trans(1,0,0)



DrawSquare()

glPushMatrix()

glScale3f(2,2,2)

glTranslate3f(1,0,0)

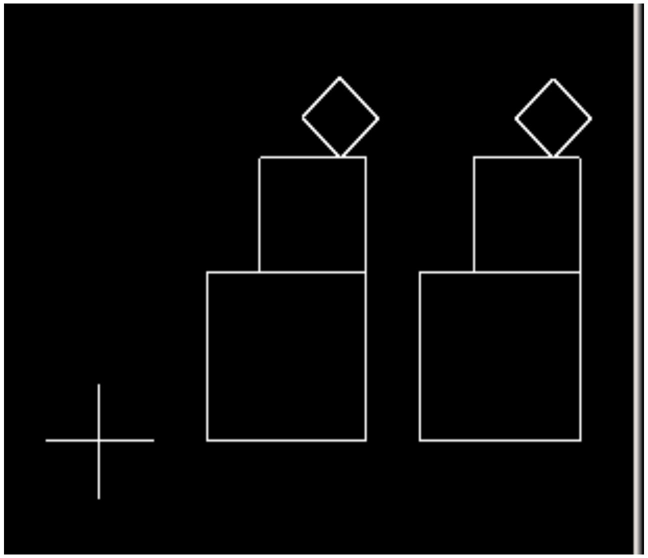
DrawSquare()

glPopMatrix()

Modularization

- drawing a scaled square
  - push/pop ensures no coord system change

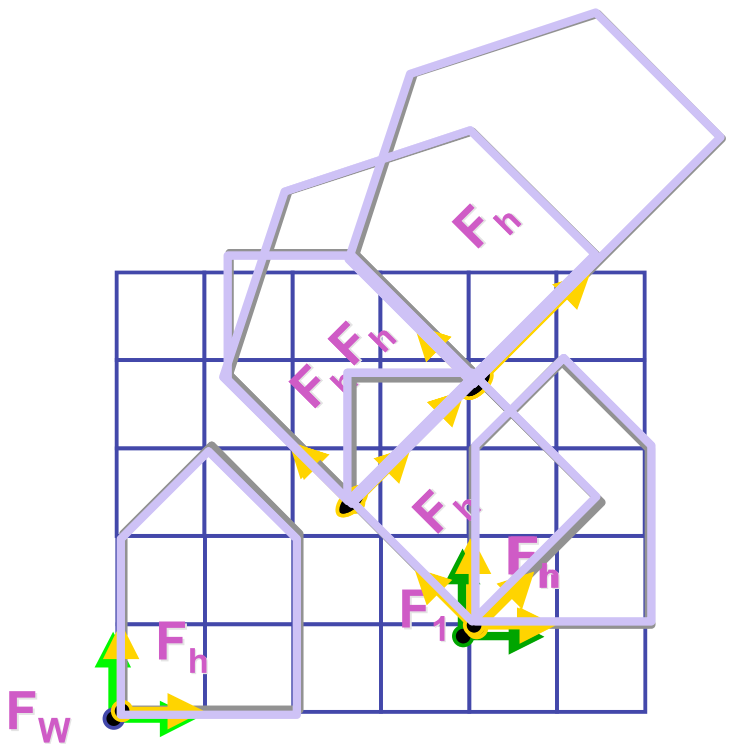
```
void drawBlock(float k) {
  glPushMatrix();

  glScalef(k,k,k);
  glBegin(GL_LINE_LOOP);
  glVertex3f(0,0,0);
  glVertex3f(1,0,0);
  glVertex3f(1,1,0);
  glVertex3f(0,1,0);
  glEnd();

  glPopMatrix();
}
```



Matrix Stacks

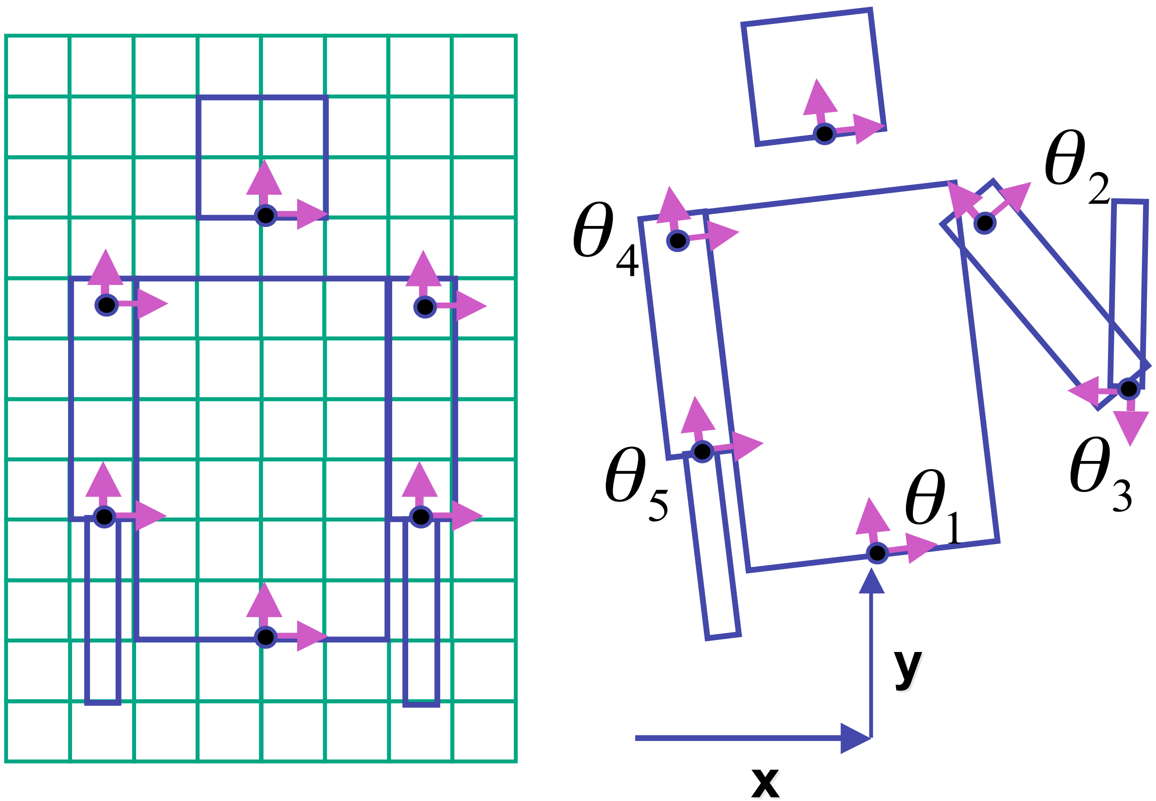
- advantages
  - no need to compute inverse matrices all the time
  - modularize changes to pipeline state
  - avoids incremental changes to coordinate systems
    - accumulation of numerical errors
- practical issues
  - in graphics hardware, depth of matrix stacks is limited
    - (typically 16 for model/view and about 4 for projective matrix)

Transformation Hierarchy Example 3




```
glLoadIdentity();
glTranslatef(4,1,0);
glPushMatrix();
glRotatef(45,0,0,1);
glTranslatef(0,2,0);
glScalef(2,1,1);
glTranslate(1,0,0);
glPopMatrix();
```

Transformation Hierarchy Example 4




```
glTranslate3f(x,y,0);
glRotatef(θ1,0,0,1);
DrawBody();
glPushMatrix();
  glTranslate3f(0,7,0);
  DrawHead();
glPopMatrix();
glPushMatrix();
  glTranslate(2.5,5.5,0);
  glRotatef(θ2,0,0,1);
  DrawUArm();
  glTranslate(0,-3.5,0);
  glRotatef(θ3,0,0,1);
  DrawLArm();
glPopMatrix();
... (draw other arm)
```

Hierarchical Modelling

- advantages
  - define object once, instantiate multiple copies
  - transformation parameters often good control knobs
  - maintain structural constraints if well-designed
- limitations
  - expressivity: not always the best controls
  - can't do closed kinematic chains
    - keep hand on hip
  - can't do other constraints
    - collision detection
      - self-intersection
      - walk through walls