University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2008

Tamara Munzner

# Transformations II

## Week 2, Fri Jan 18

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

# Assignments

# Assignments

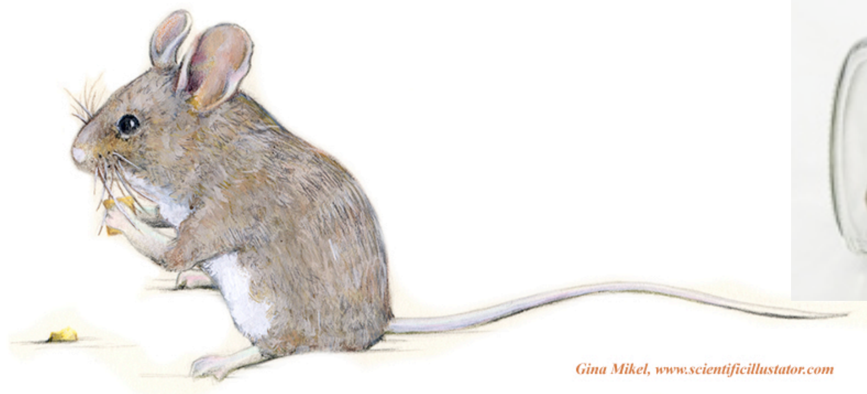
- project 1
  - out today, due 6pm Wed Feb 6
    - projects will go out before we've covered all the material
      - so you can think about it before diving in
  - build mouse out of cubes and 4x4 matrices
    - think cartoon, not beauty
  - template code gives you program shell, Makefile
    - http://www.ugrad.cs.ubc.ca/~cs314/Vjan2008/p1.tar.gz
- written homework 1
  - out Monday, due 1pm sharp Wed Feb 6
  - theoretical side of material

# Demo

- animal out of boxes and matrices

# Real Mice

http://www.scientificillustrator.com/art/wildlife/mouse.jpg



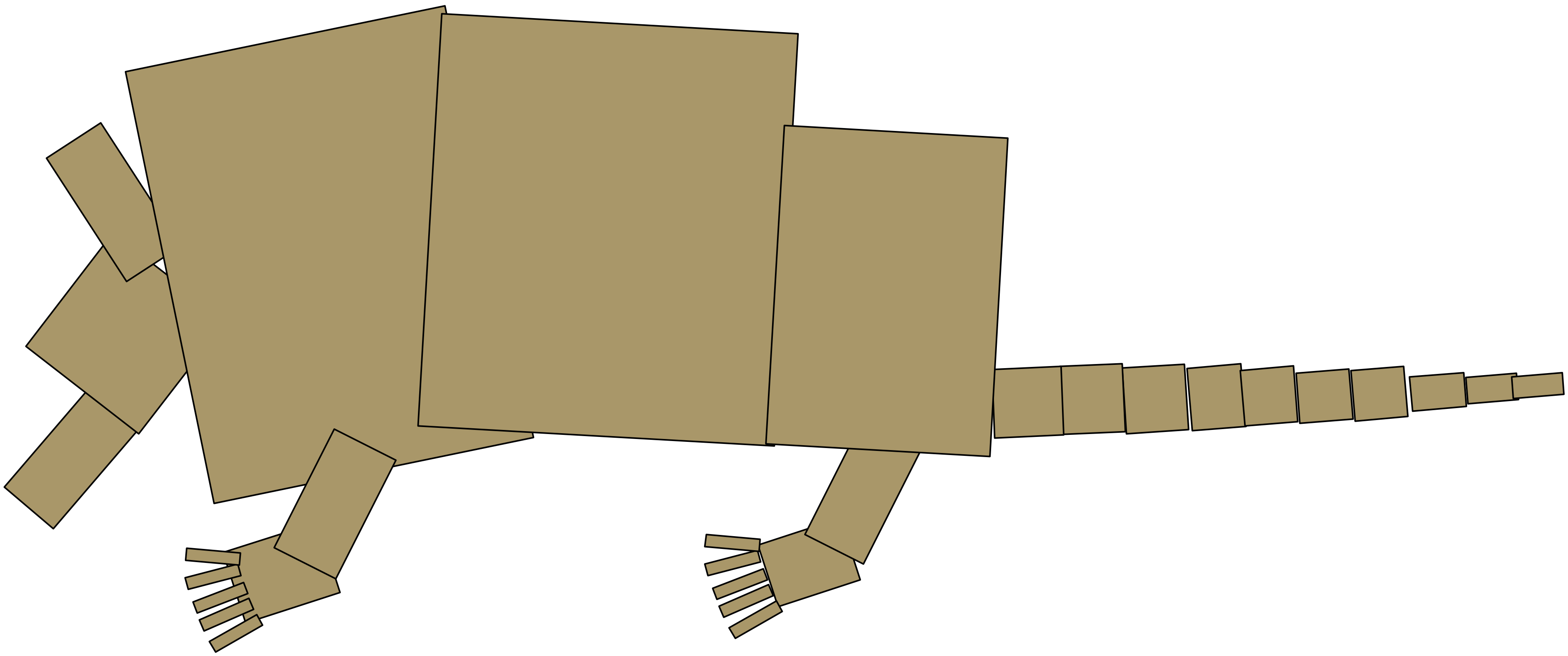

http://www.dezeen.com/wp-content/uploads/2007/10/mouse-in-a-bottle_sq.jpg

http://www.naturephoto-cz.com/photos/andera/house-mouse-13044.jpg

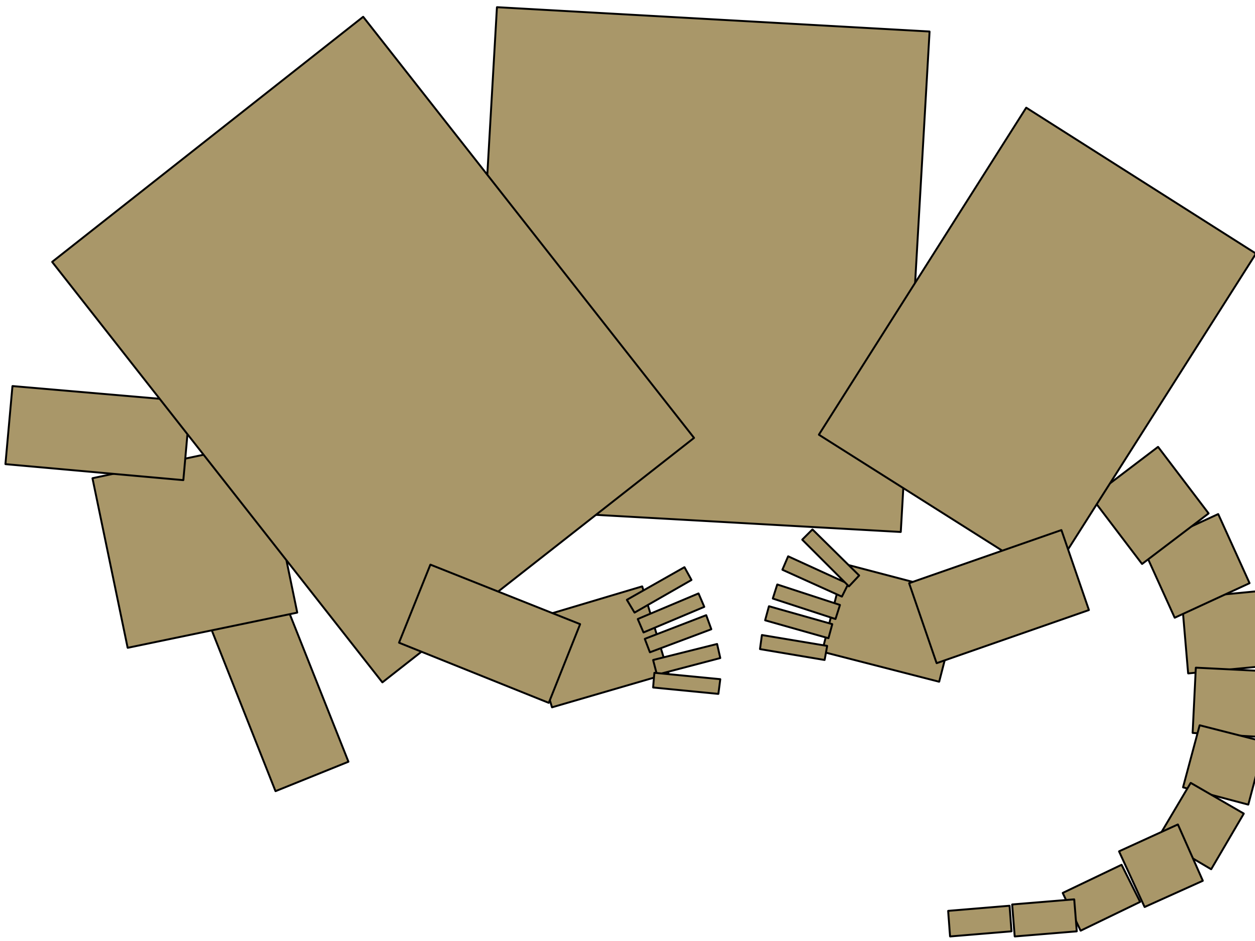

http://www.naturephoto-cz.com/photos/andera/house-mouse-15372.jpg


http://www.com.msu.edu/carcino/Resources/mouse.jpg

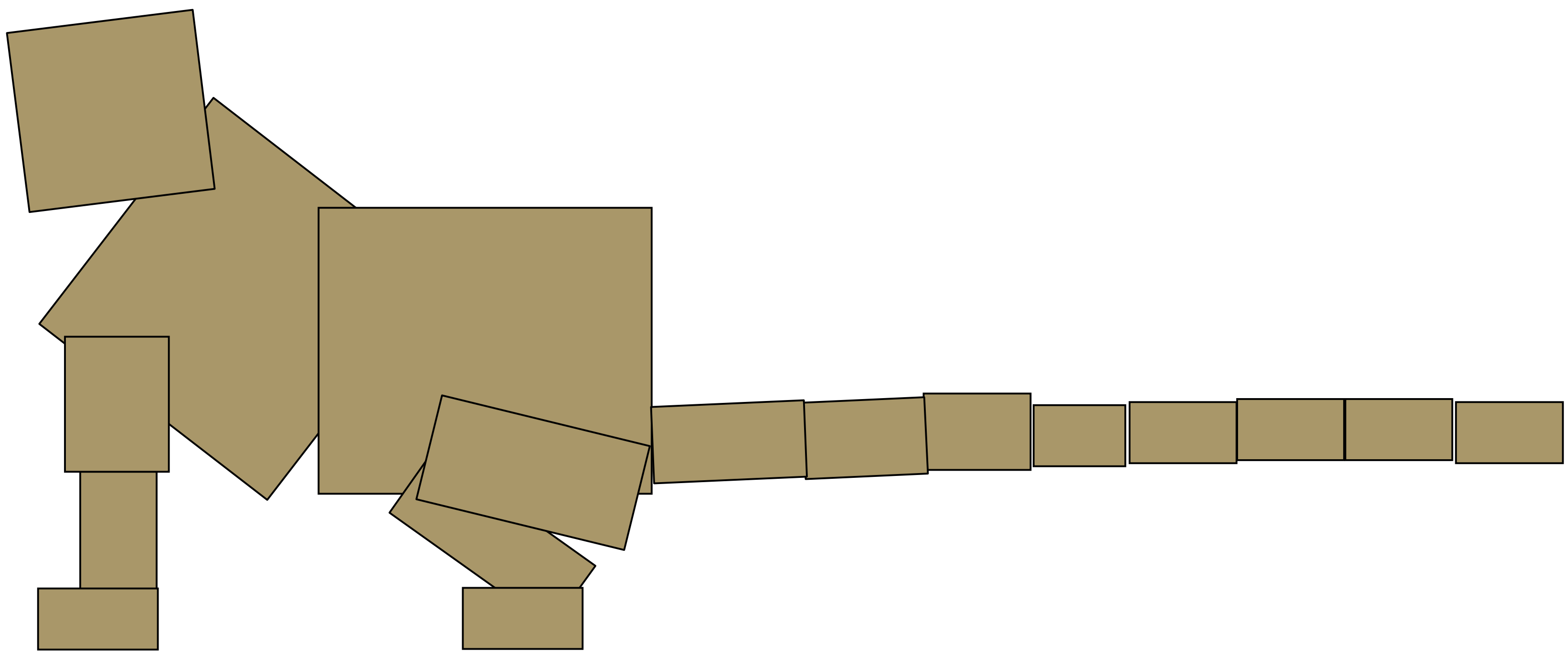
# Think Cartoon



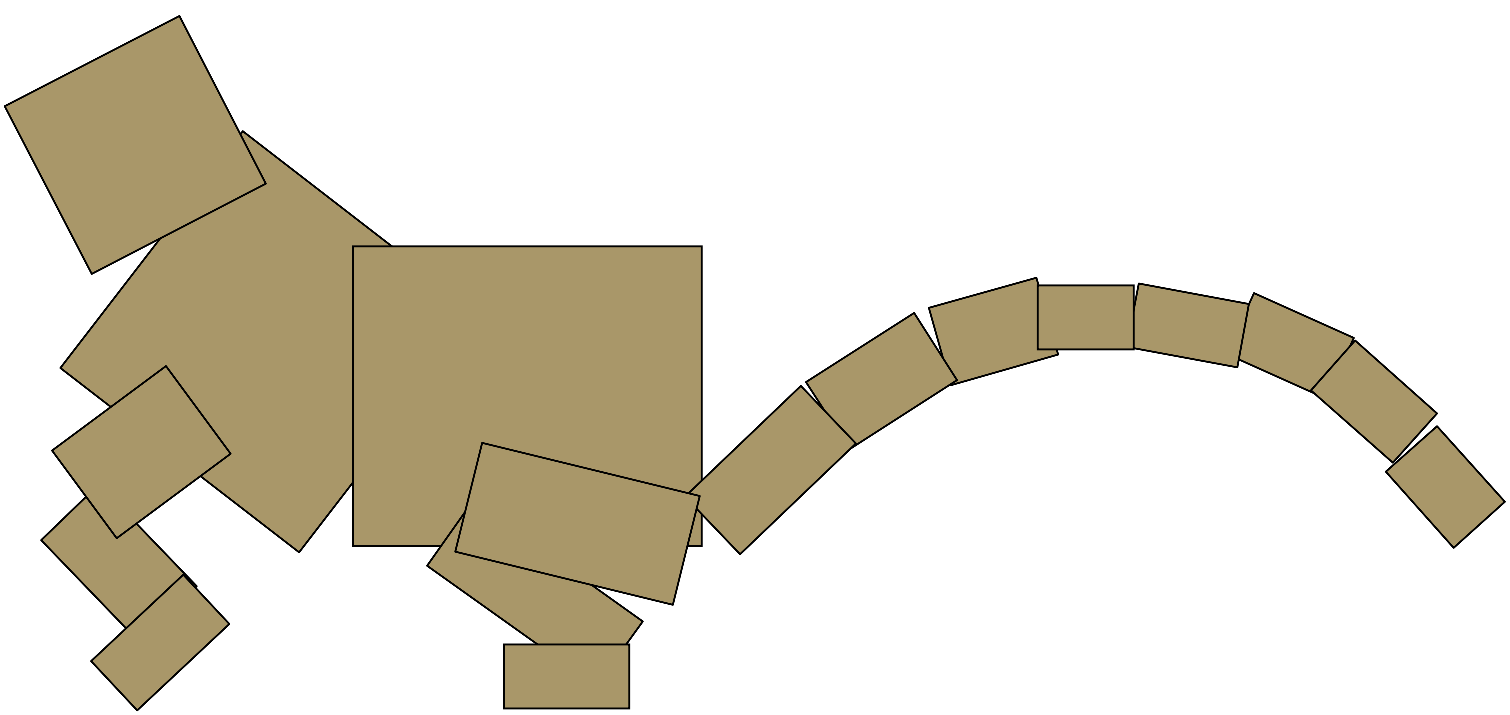
# Armadillos!



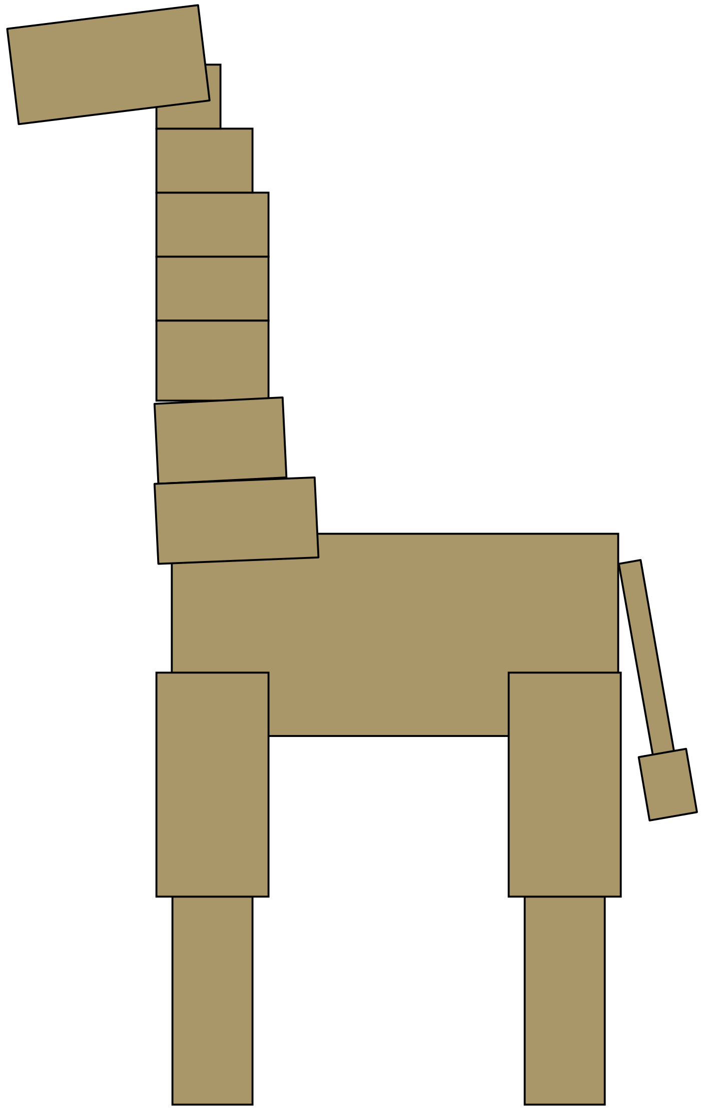
# Monkeys!



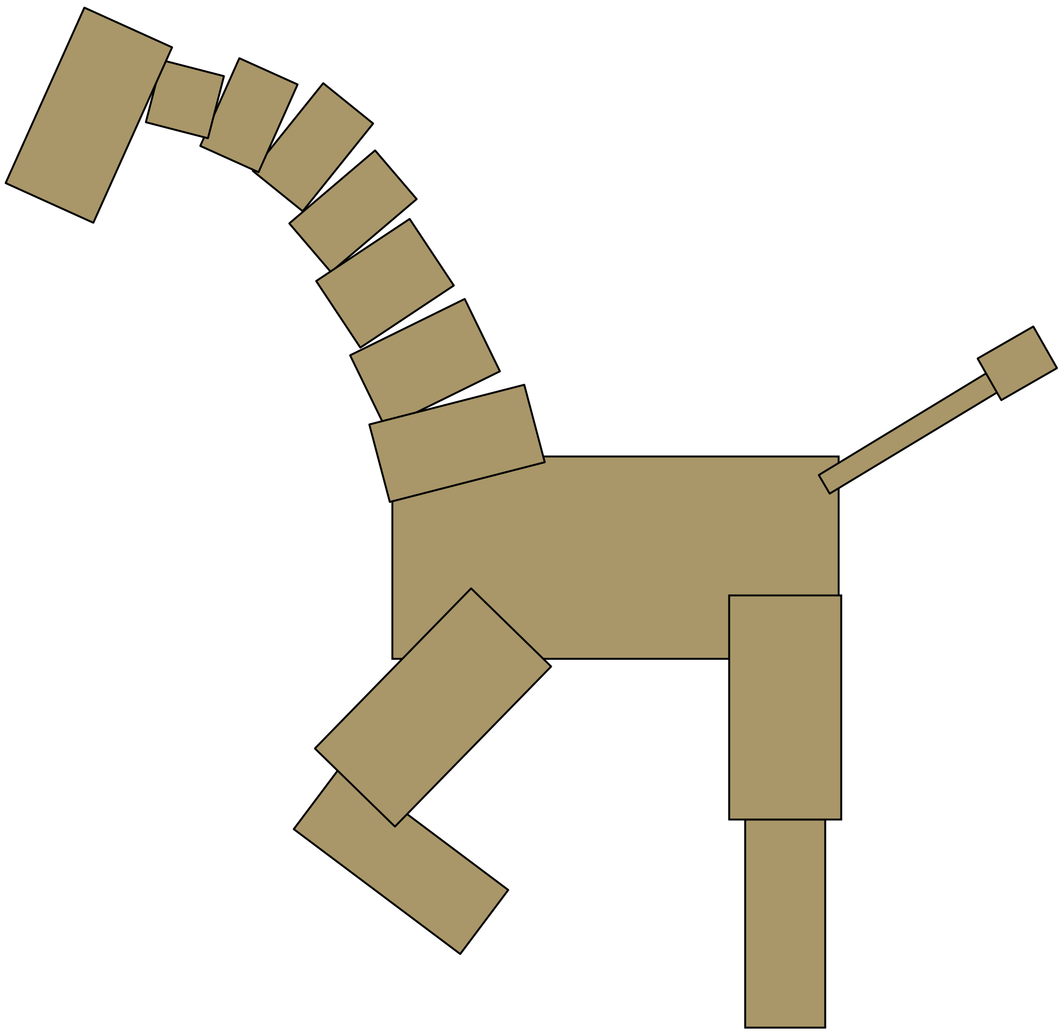
# Monkeys!



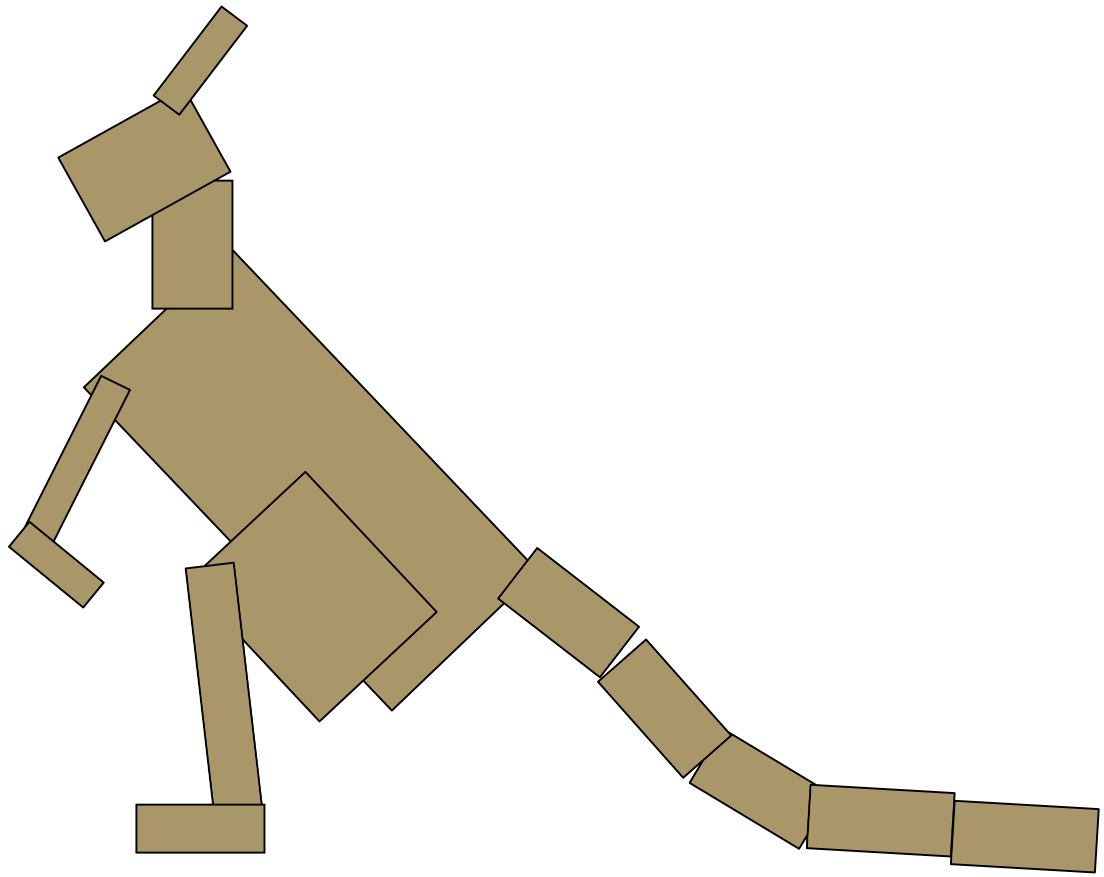
# Giraffes!



# Giraffes!



# Kangaroos!



# Project 1 Advice

- do not model everything first and only then worry about animating
- interleave modelling, animation
  - for each body part: add it, then jumpcut animate, then smooth animate
  - discover if on wrong track sooner
  - dependencies: can't get anim credit if no model
  - use body as scene graph root
- check from all camera angles

# Project 1 Advice

- finish all required parts before
  - going for extra credit
  - playing with lighting or viewing
- ok to use glRotate, glTranslate, glScale
- ok to use glutSolidCube, or build your own
  - where to put origin? your choice
    - center of object, range - .5 to +.5
    - corner of object, range 0 to 1

# Project 1 Advice

- visual debugging
  - color cube faces differently
  - colored lines sticking out of glutSolidCube faces
  - make your cubes wireframe to see inside
- thinking about transformations
  - move physical objects around
  - play with demos
    - Brown scenegraph applets

# Project 1 Advice

- smooth transition
  - change happens gradually over X frames
  - key click triggers animation
  - one way: redraw happens X times
    - linear interpolation:

      each time, param += (new-old)/30
  - or redraw happens over X seconds
    - even better, but not required

# Project 1 Advice

- transitions
  - safe to linearly interpolate parameters for glRotate/glTranslate/glScale
  - do not interpolate individual elements of 4x4 matrix!

# Style

- you can lose up to 15% for poor style
- most critical: reasonable structure
  - yes: parametrized functions
  - no: cut-and-paste with slight changes
- reasonable names (variables, functions)
- adequate commenting
  - rule of thumb: what if you had to fix a bug two years from now?
- global variables are indeed acceptable

# Version Control

- bad idea: just keep changing same file
- save off versions often
  - after got one thing to work, before you try starting something else
  - just before you do something drastic
- how?
  - not good: commenting out big blocks of code
  - a little better: save off file under new name
    - p1.almostworks.cpp, p1.fixedbug.cpp
- much better: use version control software
  - strongly recommended

# Version Control Software

- easy to browse previous work
- easy to revert if needed
- for maximum benefit, use meaningful comments to describe what you did
    - “started on tail”, “fixed head breakoff bug”, “leg code compiles but doesn’t run”
- useful when you’re working alone
- critical when you’re working together
- many choices: RCS, CVS, svn/subversion
    - all are installed on lab machines
    - svn tutorial is part of next week’s lab

# Graphical File Comparison

- installed on lab machines
  - xfdiff4 (side by side comparison)
  - xwdiff (in-place, with crossouts)
- Windows: windiff
  - http://keithdevens.com/files/windiff
- Macs: FileMerge
  - in /Developer/Applications/Utilities

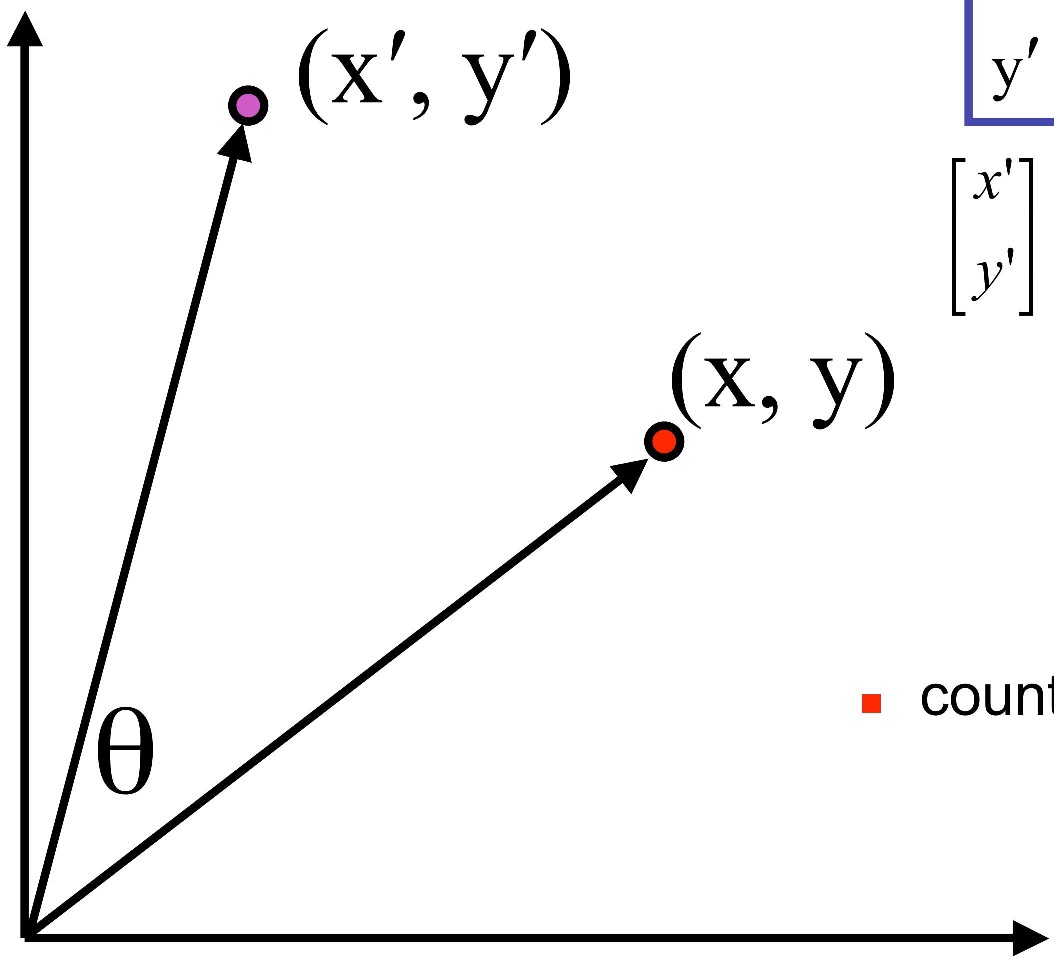
# Readings for Jan 16-25

- 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

# Review: Event-Driven Programming

- main loop not under your control
  - vs. procedural
- control flow through event callbacks
  - redraw the window now
  - key was pressed
  - mouse moved
- callback functions called from main loop when events occur
  - mouse/keyboard state setting vs. redrawing

# Review: 2D Rotation



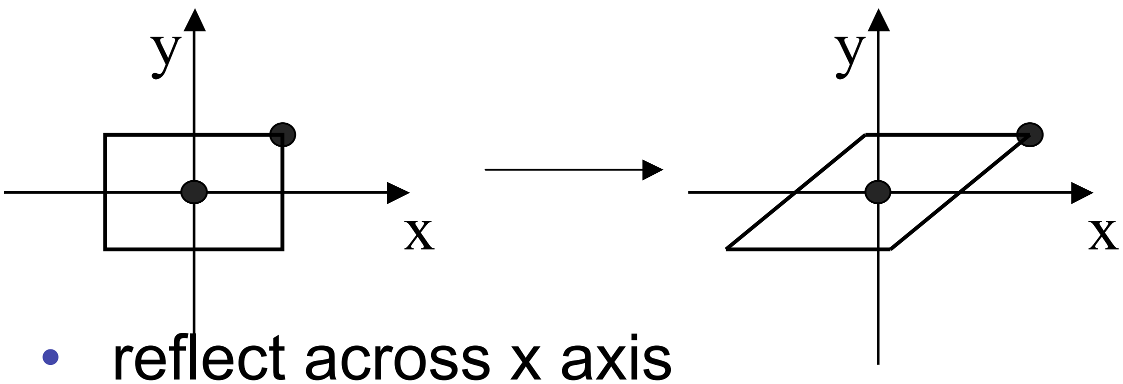

$$x' = x\,\mathbf{cos}(\theta) - y\,\mathbf{sin}(\theta)$$
$$y' = x\,\mathbf{sin}(\theta) + y\,\mathbf{cos}(\theta)$$

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

- counterclockwise, RHS

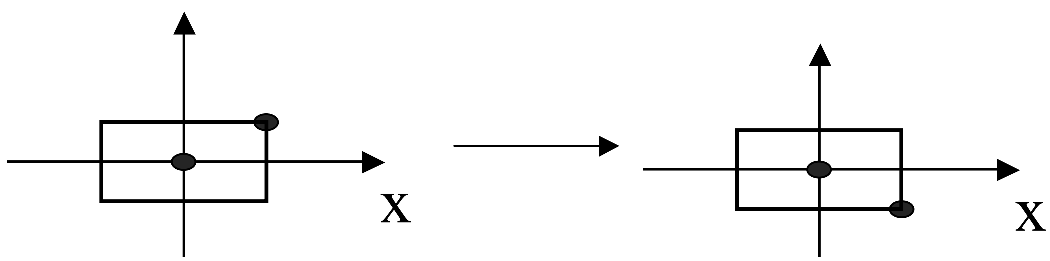
# Review: Shear, Reflection

- shear along x axis
  - push points to right in proportion to height




$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- reflect across x axis
  - mirror




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

# Review: 2D Transformations

matrix multiplication

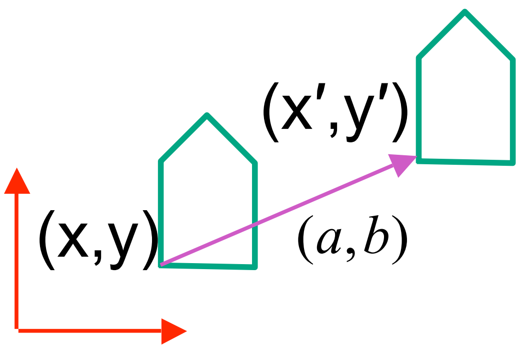
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\text{scaling matrix}} \begin{bmatrix} x \\ y \end{bmatrix}$$

*scaling matrix*

matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{\text{rotation matrix}} \begin{bmatrix} x \\ y \end{bmatrix}$$

*rotation matrix*




vector addition

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{\text{translation multiplication matrix??}} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

*translation multiplication matrix??*

# Review: Linear Transformations

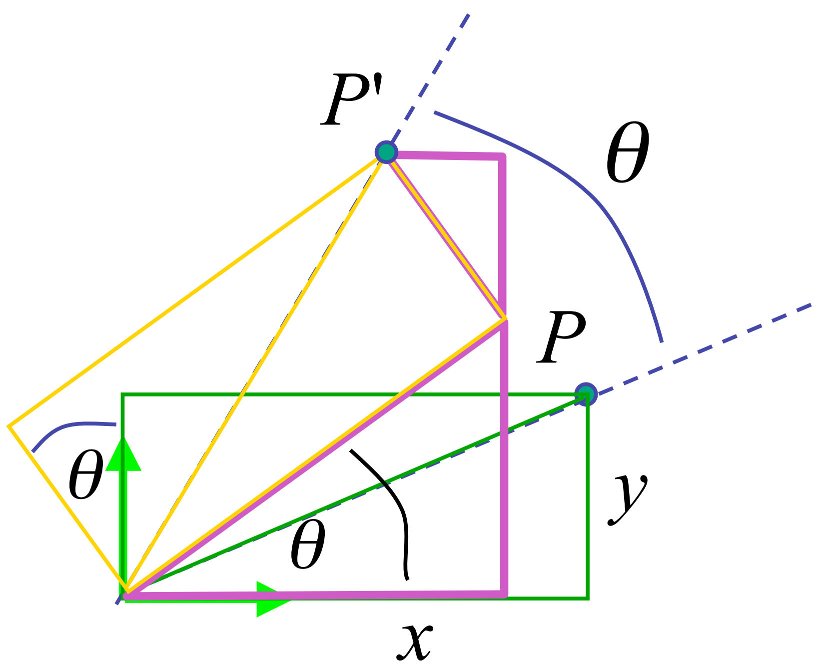
- linear transformations are combinations of
  - shear
  - scale
  - rotate
  - reflect

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

$$x' = ax + by$$
$$y' = cx + dy$$

- properties of linear transformations
  - satisifes T(*s***x**+*t***y**) = *s* T(**x**) + *t* T(**y**)
  - origin maps to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

# 3D Rotation About Z Axis




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

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

$$z' = z$$

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

- general OpenGL command
  - **glRotatef(angle,x,y,z);**
- rotate in z
  - **glRotatef(angle,0,0,1);**

# 3D Rotation in X, Y

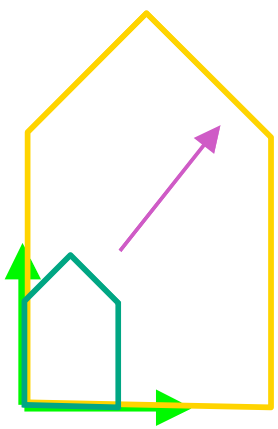
around x axis: **glRotatef(angle,1,0,0);**

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

around y axis: **glRotatef(angle,0,1,0);**

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

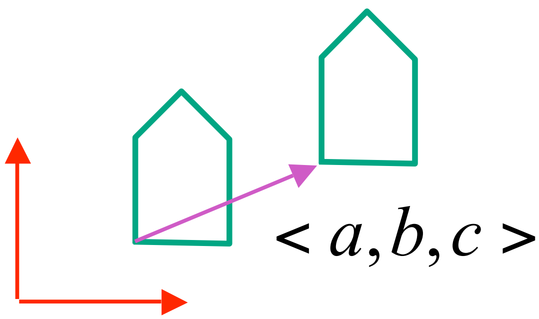
# 3D Scaling



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

**glScalef(a,b,c);**

# 3D Translation




$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

**glTranslatef(a,b,c);**

# 3D Shear

- general shear

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

  - to avoid ambiguity, always say "shear along <axis> in direction of <axis>"

$$shearAlongXinDirectionOfY(h) = \begin{bmatrix} 1 & h & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$shearAlongXinDirectionOfZ(h) = \begin{bmatrix} 1 & 0 & h & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$shearAlongYinDirectionOfX(h) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ h & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$shearAlongYinDirectionOfZ(h) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & h & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$shearAlongZinDirectionOfX(h) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ h & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$shearAlongZinDirectionOfY(h) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & h & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Summary: Transformations

**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)**

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

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

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

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

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

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

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

# Undoing Transformations: Inverses

$$\mathbf{T}(x,y,z)^{-1} = \mathbf{T}(-x,-y,-z)$$

$$\mathbf{T}(x,y,z)\,\mathbf{T}(-x,-y,-z) = \mathbf{I}$$

$$\mathbf{R}(z,\theta)^{-1} = \mathbf{R}(z,-\theta) = \mathbf{R}^{\mathrm{T}}(z,\theta)$$ **(R is orthogonal)**

$$\mathbf{R}(z,\theta)\,\mathbf{R}(z,-\theta) = \mathbf{I}$$

$$\mathbf{S}(sx,sy,sz)^{-1} = \mathbf{S}(\frac{1}{sx},\frac{1}{sy},\frac{1}{sz})$$

$$\mathbf{S}(sx,sy,sz)\mathbf{S}(\frac{1}{sx},\frac{1}{sy},\frac{1}{sz}) = \mathbf{I}$$

# Composing Transformations

# Composing Transformations

- translation

$$T1 = T(dx_1, dy_1) = \begin{bmatrix} 1 & & & dx_1 \\ & 1 & & dy_1 \\ & & 1 & \\ & & & 1 \end{bmatrix} \qquad T2 = T(dx_2, dy_2) = \begin{bmatrix} 1 & & & dx_2 \\ & 1 & & dy_2 \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

$$P'' = T2 \bullet P' = T2 \bullet [T1 \bullet P] = [T2 \bullet T1] \bullet P, where$$

$$T2 \bullet T1 = \begin{bmatrix} 1 & & & dx_1 + dx_2 \\ & 1 & & dy_1 + dy_2 \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

**so translations add**

# Composing Transformations

- scaling

$$S2 \bullet S1 = \begin{bmatrix} sx_1 * dx_2 & & & \\ & sy_1 * sy_2 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

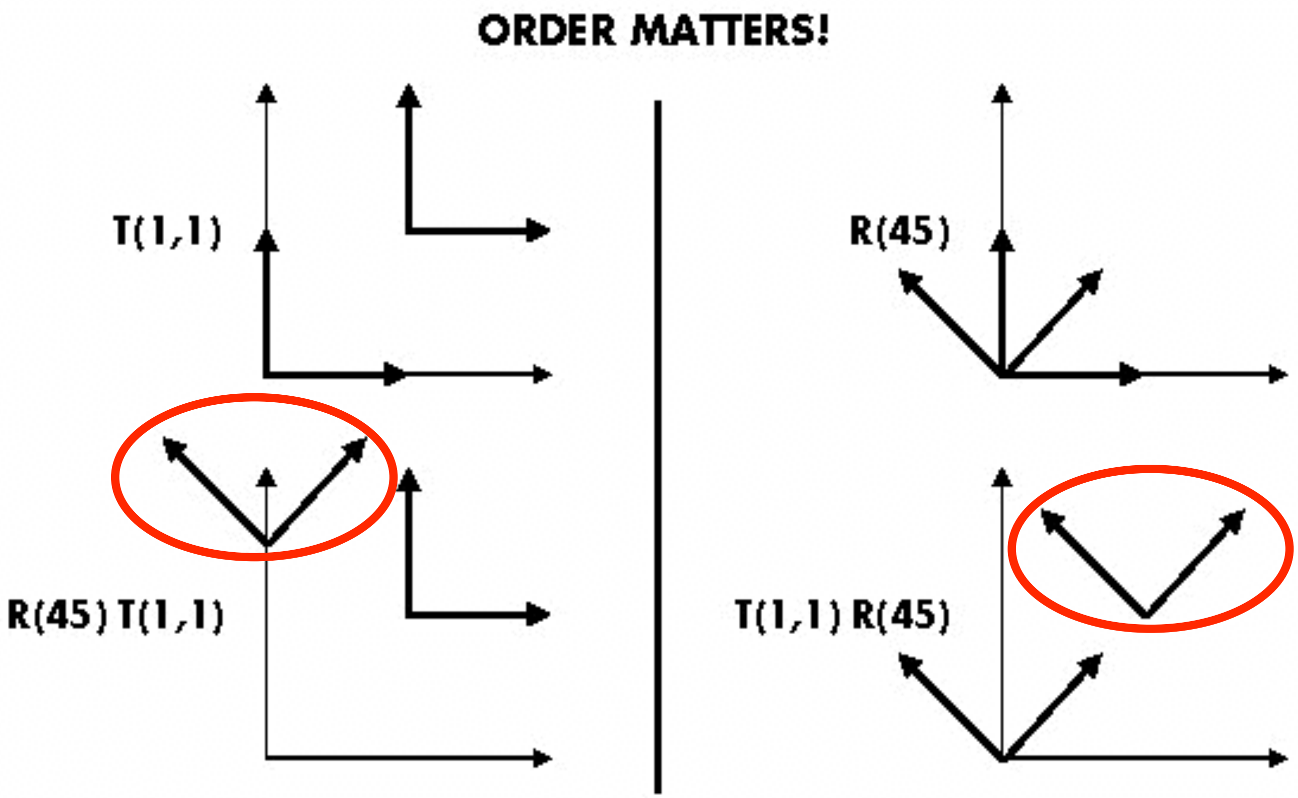
**so scales multiply**

- rotation

$$R2 \bullet R1 = \begin{bmatrix} \cos(\theta 1 + \theta 2) & -\sin(\theta 1 + \theta 2) & & \\ \sin(\theta 1 + \theta 2) & \cos(\theta 1 + \theta 2) & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

**so rotations add**

# Composing Transformations

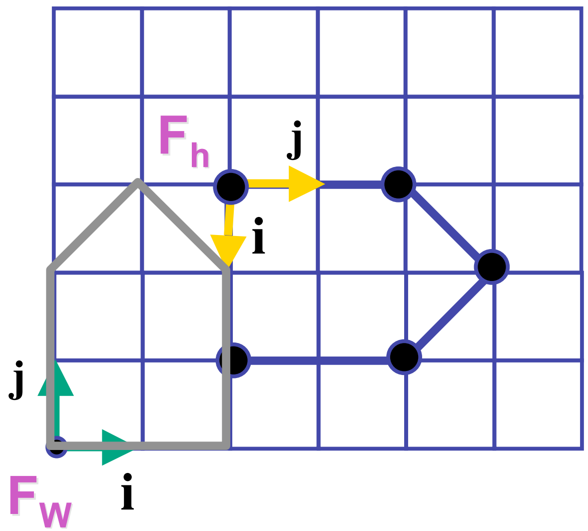



**Ta Tb = Tb Ta, but Ra Rb != Rb Ra and Ta Rb != Rb Ta**

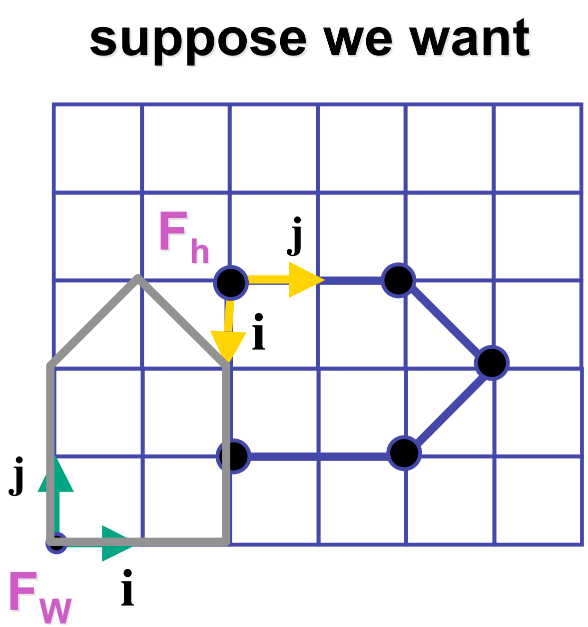
- translations commute
- rotations around same axis commute
- rotations around different axes do not commute
- rotations and translations do not commute

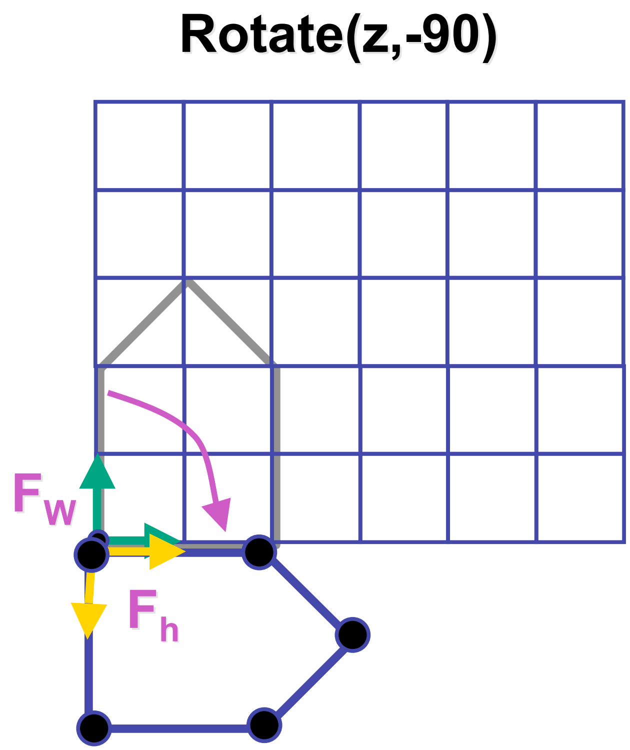
# Composing Transformations

**suppose we want**



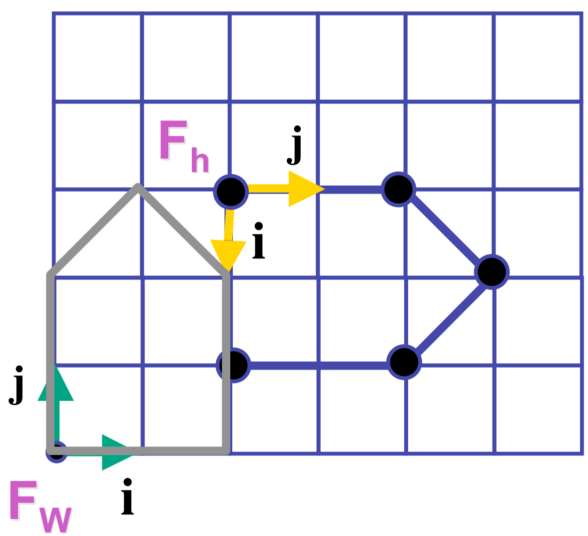
# Composing Transformations

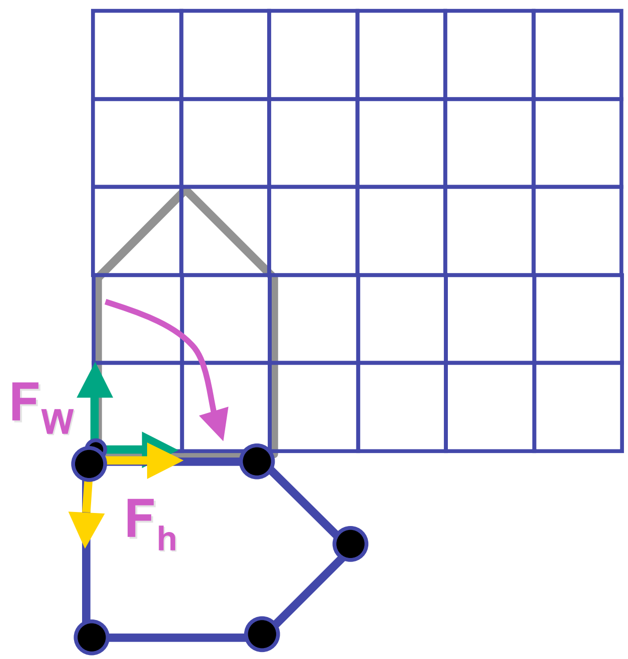


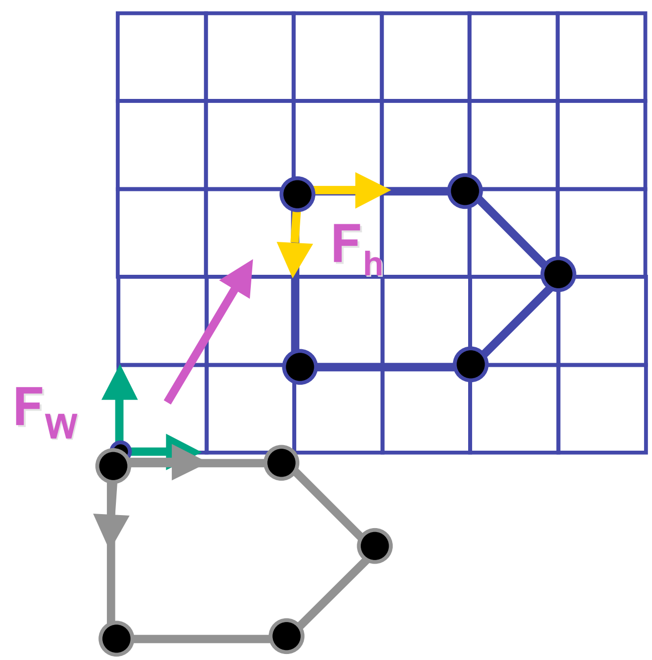

$$\mathbf{p}' = \mathbf{R}(z, -90)\,\mathbf{p}$$

# Composing Transformations



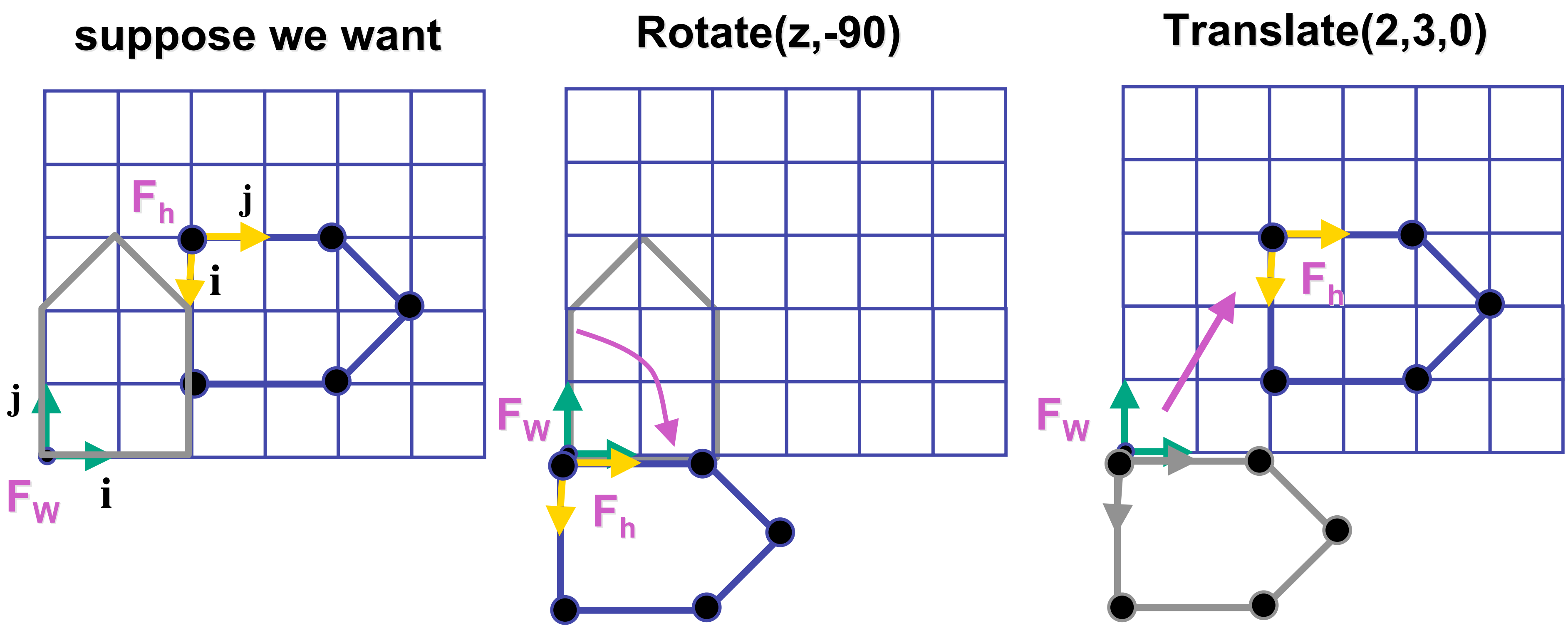




$$\mathbf{p}' = \mathbf{R}(z, -90)\,\mathbf{p}$$




$$\mathbf{p}'' = \mathbf{T}(2,3,0)\,\mathbf{p}'$$

# Composing Transformations




$$\mathbf{p}' = \mathbf{R}(z,-90)\,\mathbf{p}$$

$$\mathbf{p}'' = \mathbf{T}(2,3,0)\,\mathbf{p}'$$

$$\mathbf{p}'' = \mathbf{T}(2,3,0)\,\mathbf{R}(z,-90)\mathbf{p} = \mathbf{TRp}$$

# 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
    - interpret operations wrt local coordinates
    - changing coordinate system

# Composing Transformations

$$\mathbf{p}' = \mathbf{TRp}$$

- 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

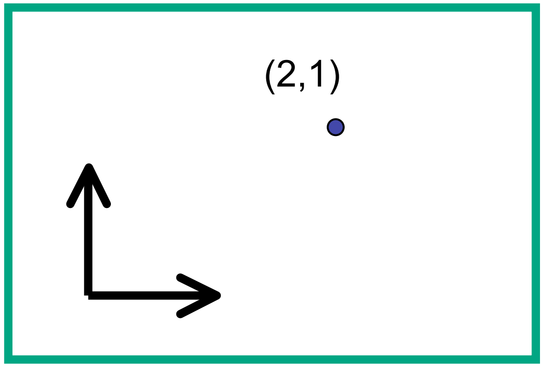
# Composing Transformations

$$\mathbf{p}' = \mathbf{TRp}$$

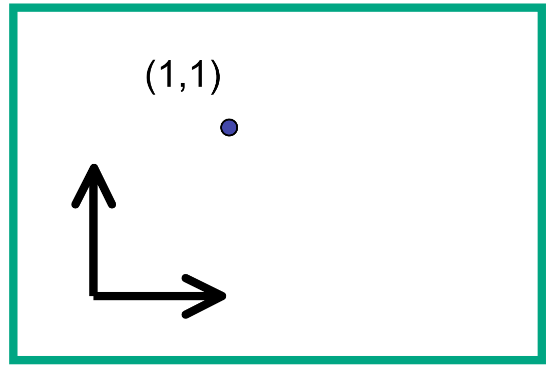
- which direction to read?
  - right to left
    - interpret operations wrt fixed coordinates
    - moving object
  - left to right **OpenGL pipeline ordering!**
    - interpret operations wrt local coordinates
    - changing coordinate system
    - OpenGL updates current matrix with postmultiply
      - glTranslatef(2,3,0);
      - glRotatef(-90,0,0,1);
      - glVertexf(1,1,1);
    - specify vector last, in final coordinate system
    - first matrix to affect it is specified second-to-last

# Interpreting Transformations

translate by (-1,0)

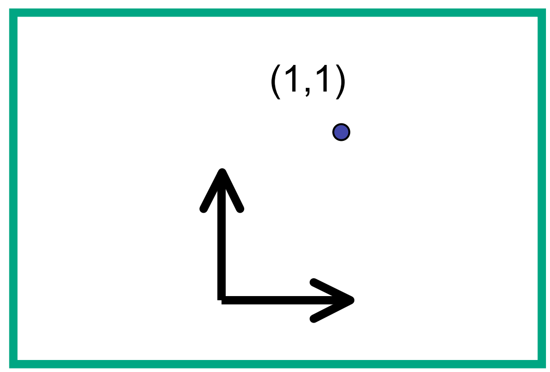



moving object




intuitive?

changing coordinate system



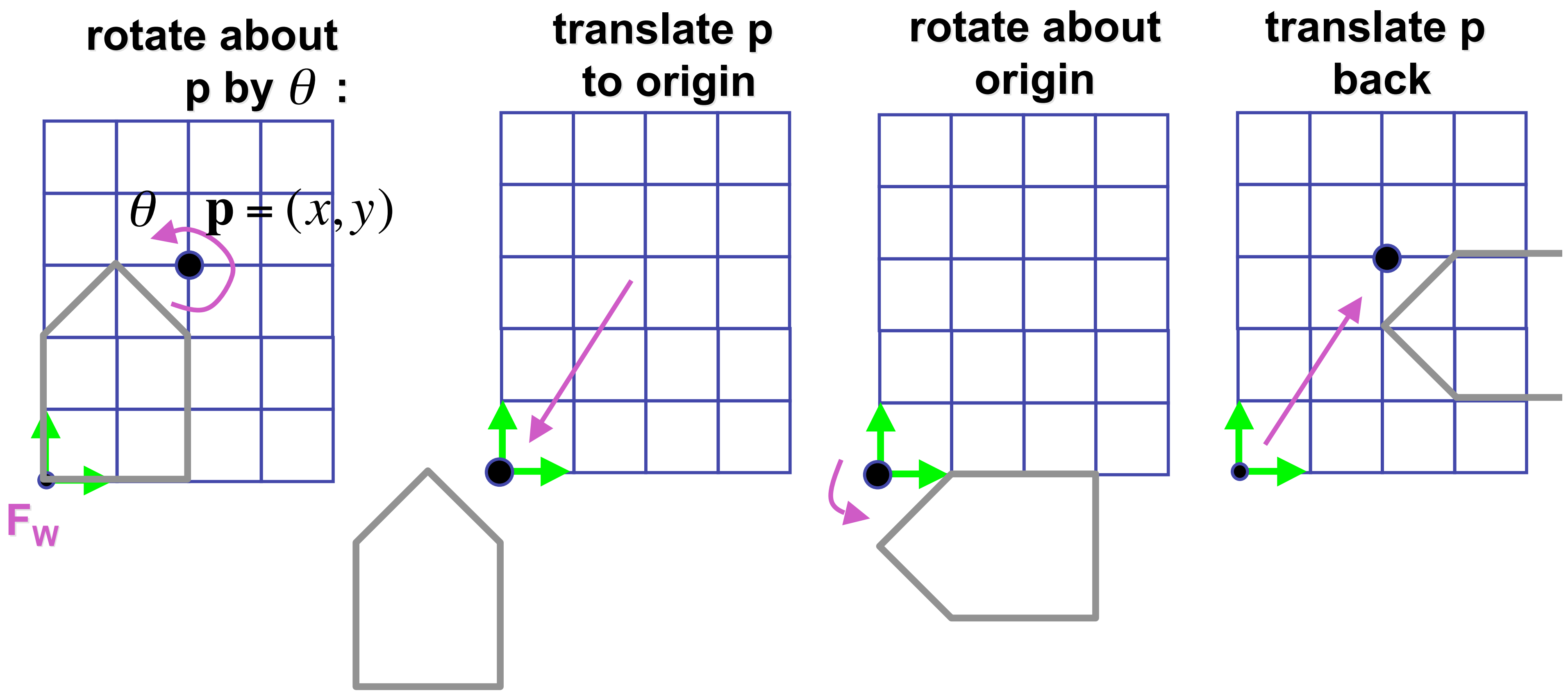

OpenGL

- same relative position between object and basis vectors

# 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

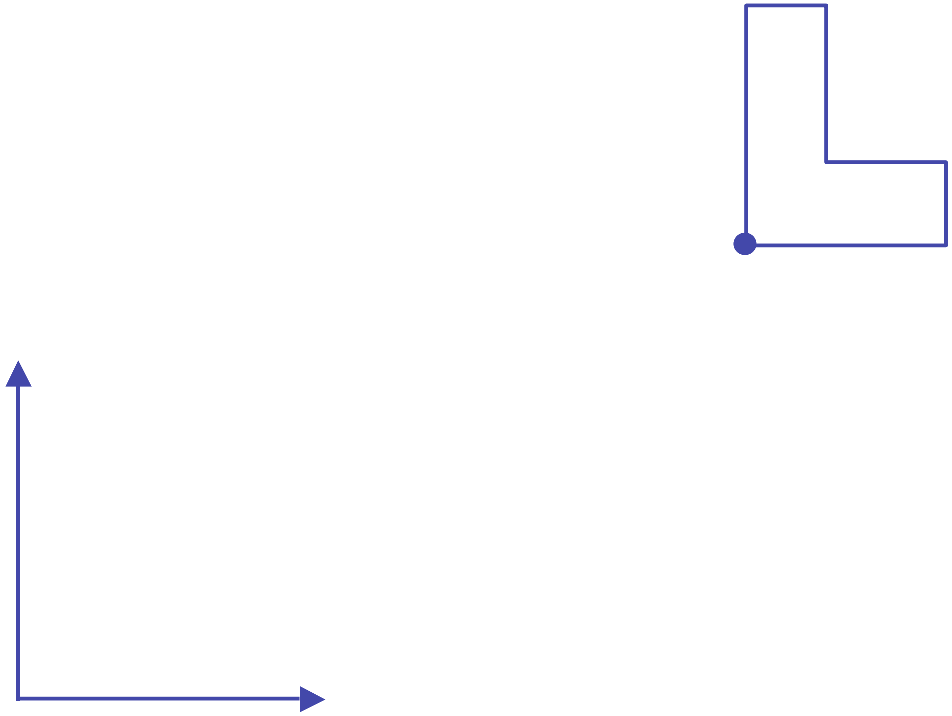
# Rotation About a Point: Moving Object




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

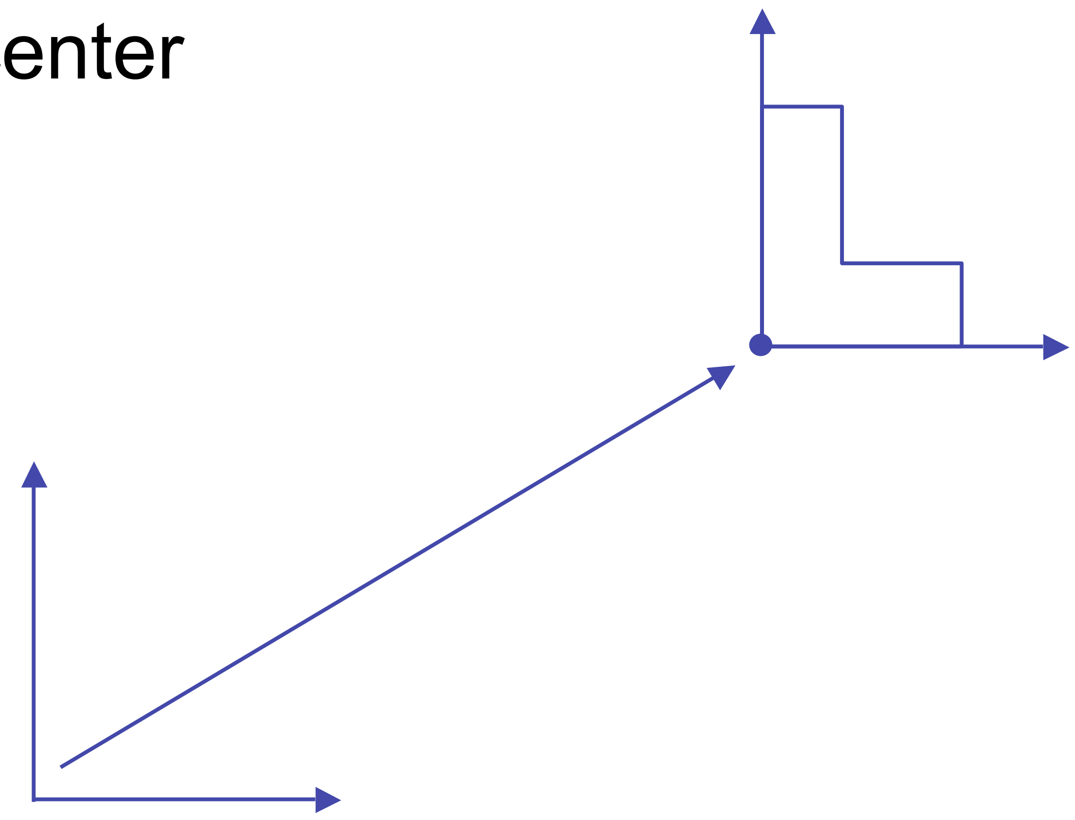
# Rotation: Changing Coordinate Systems

- same example: rotation around arbitrary center



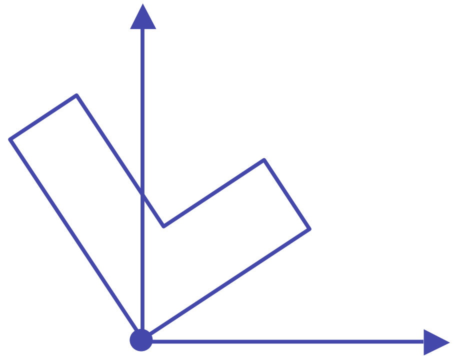
# Rotation: Changing Coordinate Systems

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



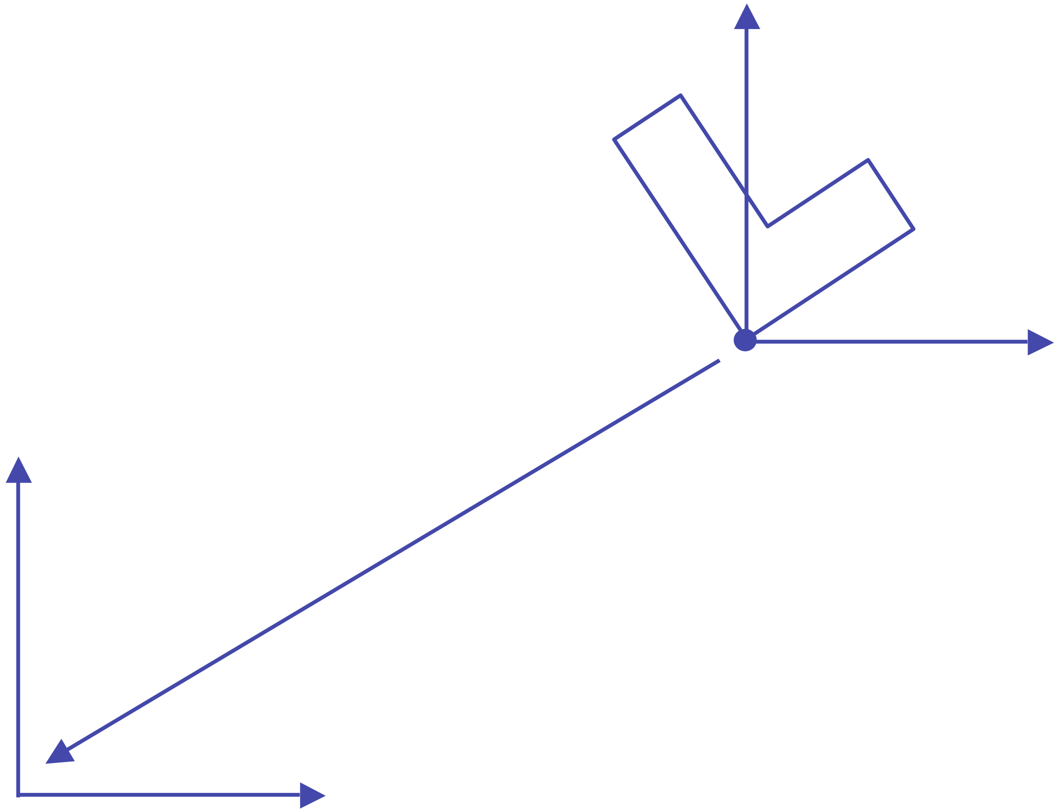
# Rotation: Changing Coordinate Systems

- rotation around arbitrary center
  - step 2: perform rotation



# Rotation: Changing Coordinate Systems

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



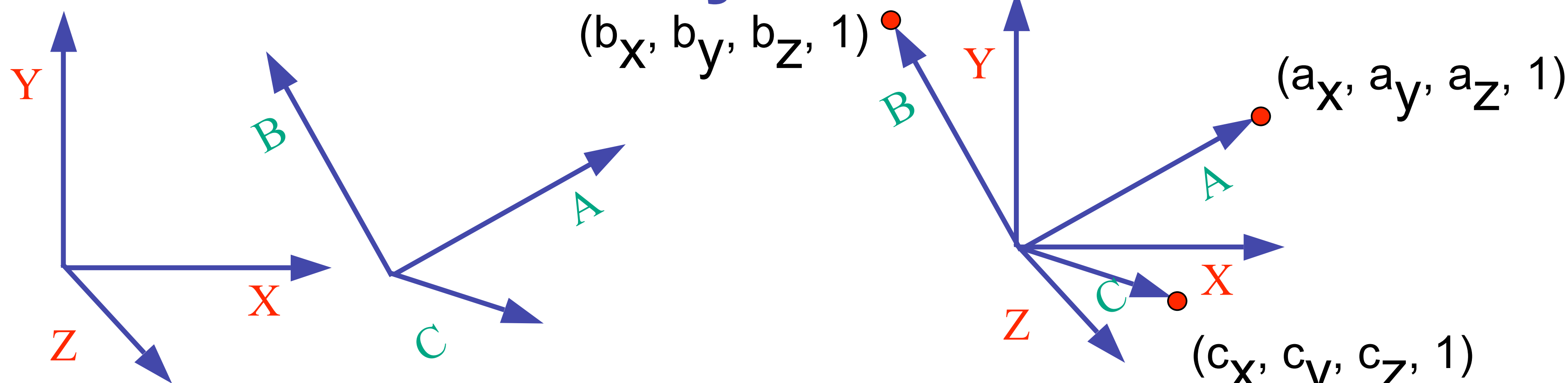
# General Transform Composition

- 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*:

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