University of British Columbia
CPSC 314 Computer Graphics
May-June 2005

Tamara Munzner

Transformations I, II, III

Week 1, Thu May 12

http://www.ugrad.cs.ubc.ca/~cs314/Vmay2005

Reading

- FCG Chap 5 (except 5.1.6, 5.3.1)
- FCG pages 224-225
- RB Chap **Viewing**:
  - Sect. Viewing and Modeling Transforms until Viewing Transformations
  - Sect. Examples of Composing Several Transformations through Building an Articulated Robot Arm
- RB Appendix **Homogeneous Coordinates and Transformation Matrices**
  - until Perspective Projection
- RB Chapter Display Lists
  - (it’s short)

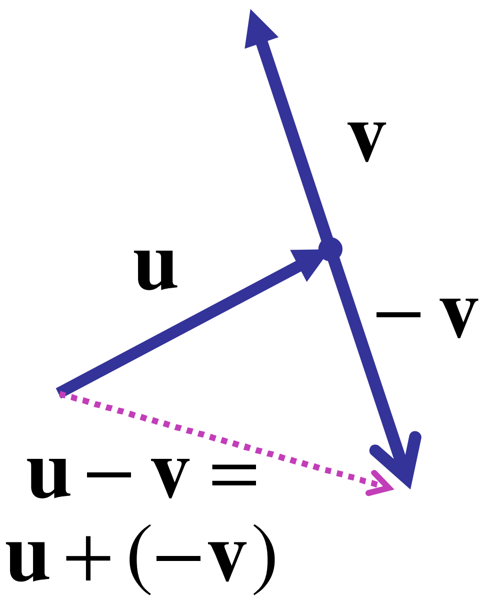
Textbook Errata

- list at http://www.cs.utah.edu/~shirley/fcg/errata
  - math review: also p 48
    - a x (b x c) != (a x b) x c
  - transforms: p 91
    - should halve x (not y) in Fig 5.10
  - transforms: p 106
    - second line of matrices: [x_p, y_p, 1]

Correction: Vector-Vector Subtraction

- subtract: vector - vector = vector

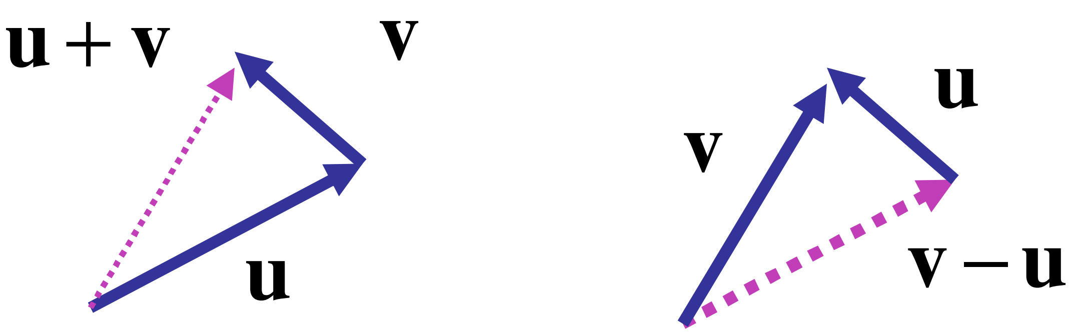
$$\mathbf{u}-\mathbf{v}=\begin{bmatrix} u_1-v_1 \\ u_2-v_2 \\ u_3-v_3 \end{bmatrix}$$



$$(3,2)-(6,4)=(-3,-2)$$

$$(2,5,1)-(3,1,-1)=(-1,\boxed{2},0)$$

argument reversal



Correction: Vector-Vector Multiplication

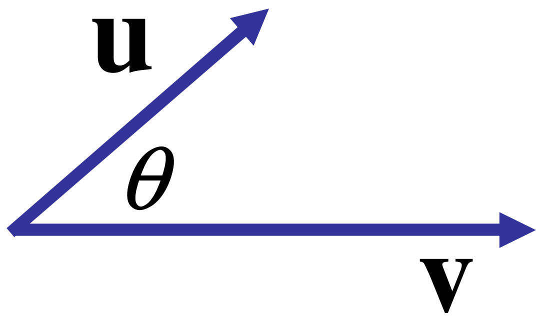
- multiply: vector * vector = scalar
- dot product, aka inner product

$$\mathbf{u} \bullet \mathbf{v}$$

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \bullet \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = (u_1 * v_1) + (u_2 * v_2) + (u_3 * v_3)$$

$$\mathbf{u} \bullet \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos\theta$$

- geometric interpretation
  - lengths, angles
  - can find angle between two vectors



Correction: Matrix Multiplication

- can only multiply (n,k) by (k,m)
  number of left cols = number of right rows
  - legal

$$\begin{bmatrix} a & b & c \\ e & f & g \end{bmatrix} \begin{bmatrix} h & i \\ j & k \\ l & m \end{bmatrix}$$

  - undefined

$$\begin{bmatrix} a & b & c \\ e & f & g \\ o & p & q \end{bmatrix} \begin{bmatrix} h & i \\ j & k \end{bmatrix}$$

Correction: Matrices and Linear Systems

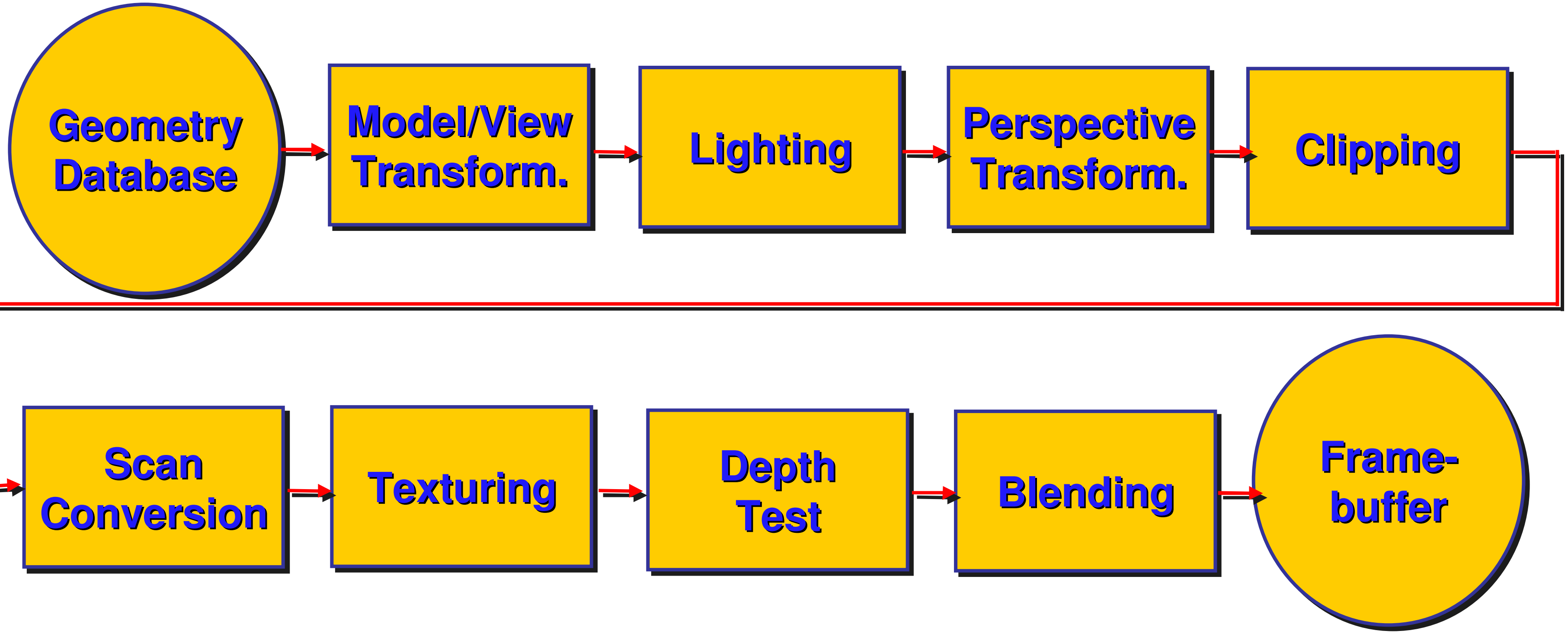
- linear system of n equations, n unknowns

$$3x + 7y + 2z = 4$$
$$2x - 4y - 3z = -1$$
$$5x + 2y + z = 1$$

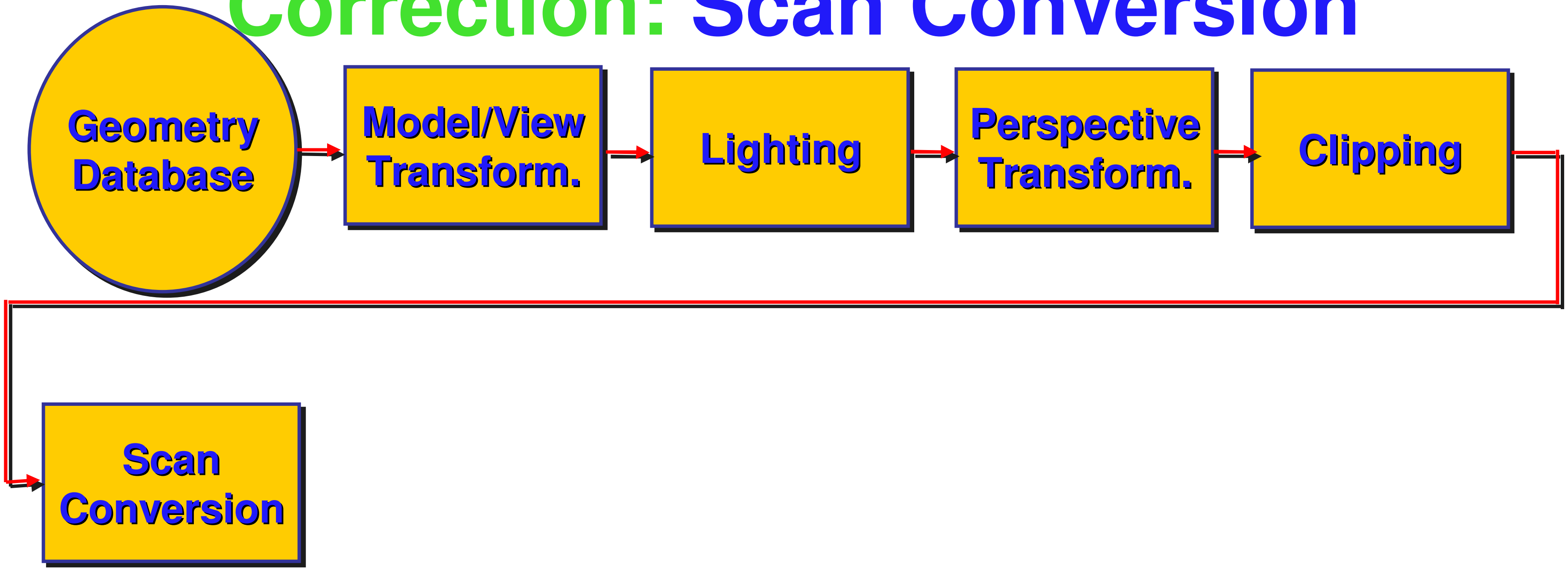
- matrix form **Ax**=**b**

$$\begin{bmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \\ \boxed{1} \end{bmatrix}$$

Review: Rendering Pipeline

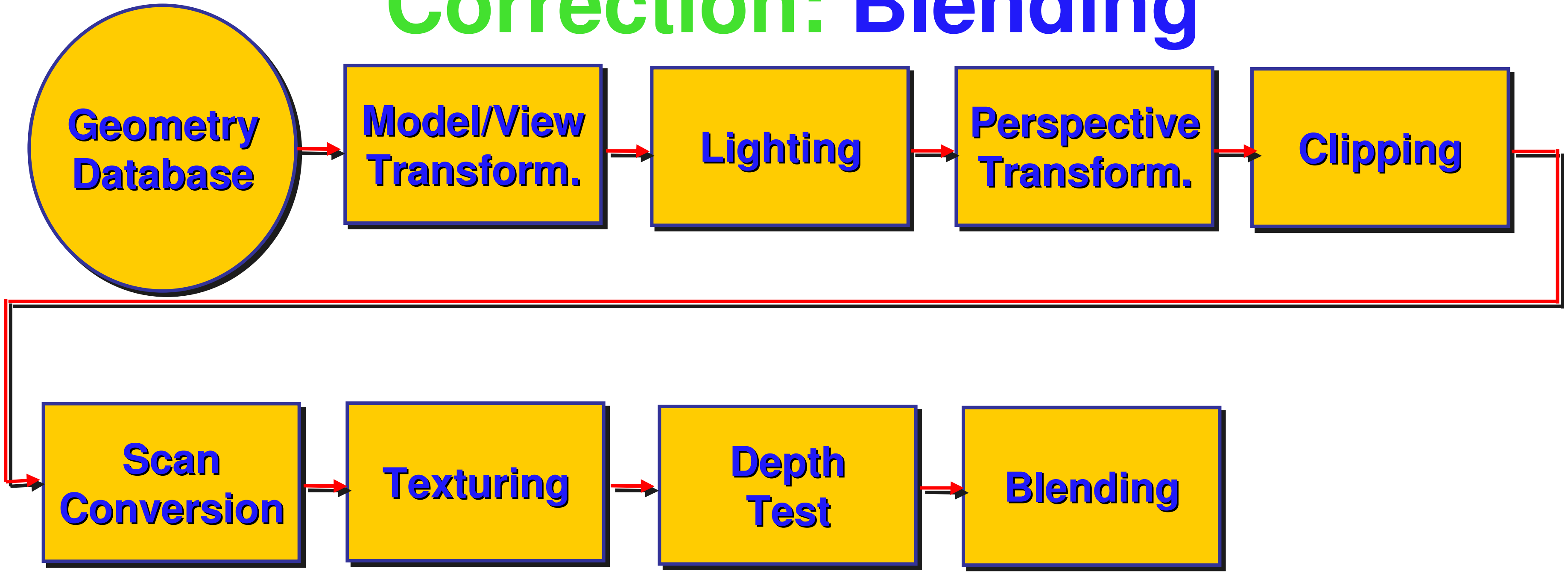


Correction: Scan Conversion



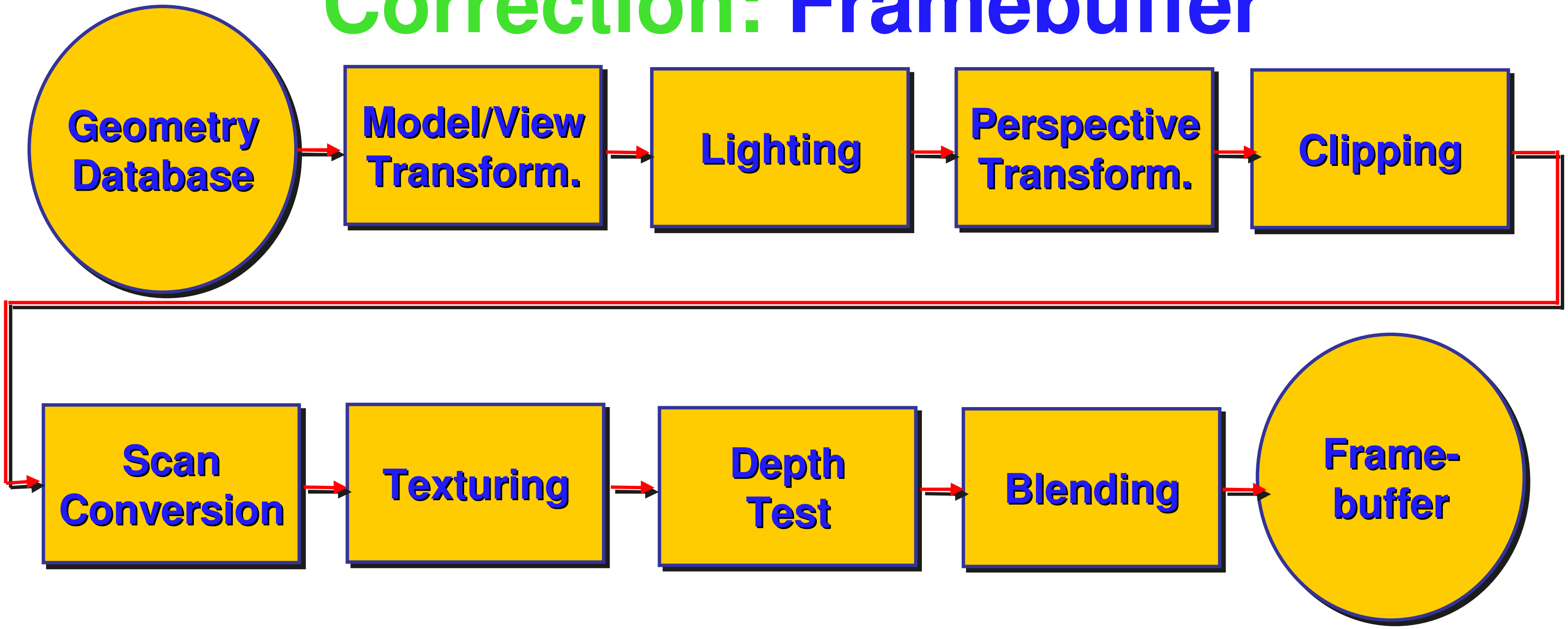
- scan conversion
  - turn 2D drawing primitives (lines, polygons etc.) into individual pixels (discretizing/sampling)
  - interpolate color across primitive
  - generate discrete fragments

Correction: Blending



- blending
  - final image: write fragments to pixels
  - draw from farthest to nearest
  - no blending – replace previous color
  - blending: combine new & old values with arithmetic operations

Correction: Framebuffer



- framebuffer
  - video memory on graphics board that holds image
  - double-buffering: two separate buffers
    - draw into one while displaying other, then swap
    - allows smooth animation, instead of flickering

Review: OpenGL

- pipeline processing, set state as needed

```
void display()
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 1.0, 0.0);
  glBegin(GL_POLYGON);
    glVertex3f(0.25, 0.25, -0.5);
    glVertex3f(0.75, 0.25, -0.5);
    glVertex3f(0.75, 0.75, -0.5);
    glVertex3f(0.25, 0.75, -0.5);
  glEnd();
  glFlush();
}
```

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

Transformations

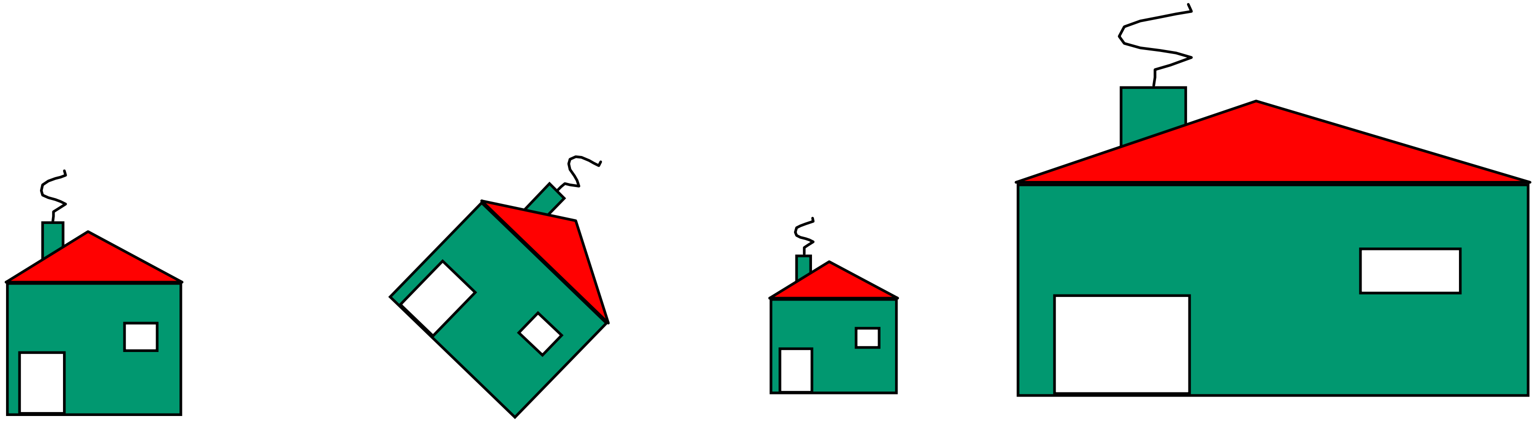
Overview

- 2D Transformations
- Homogeneous Coordinates
- 3D Transformations
- Composing Transformations
- Transformation Hierarchies
- Display Lists
- Transforming Normals
- Assignments

Transformations

- transforming an object = transforming all its points

- transforming a polygon = transforming its vertices



Matrix Representation

- represent 2D transformation with matrix
  - multiply matrix by column vector $\Leftrightarrow$ apply transformation to point

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{aligned} x' &= ax + by \\ y' &= cx + dy \end{aligned}$$

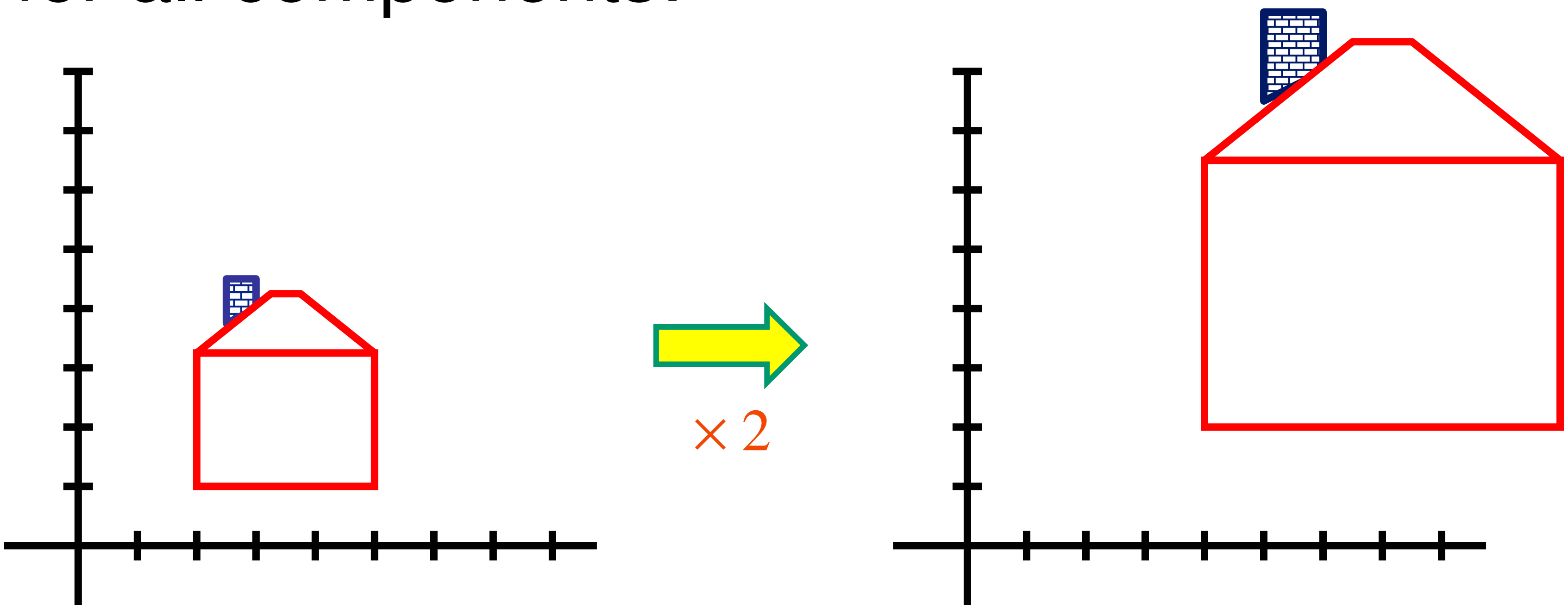
- transformations combined by multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} d & e \\ f & g \end{bmatrix} \begin{bmatrix} h & i \\ j & k \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

  - matrices are efficient, convenient way to represent sequence of transformations!

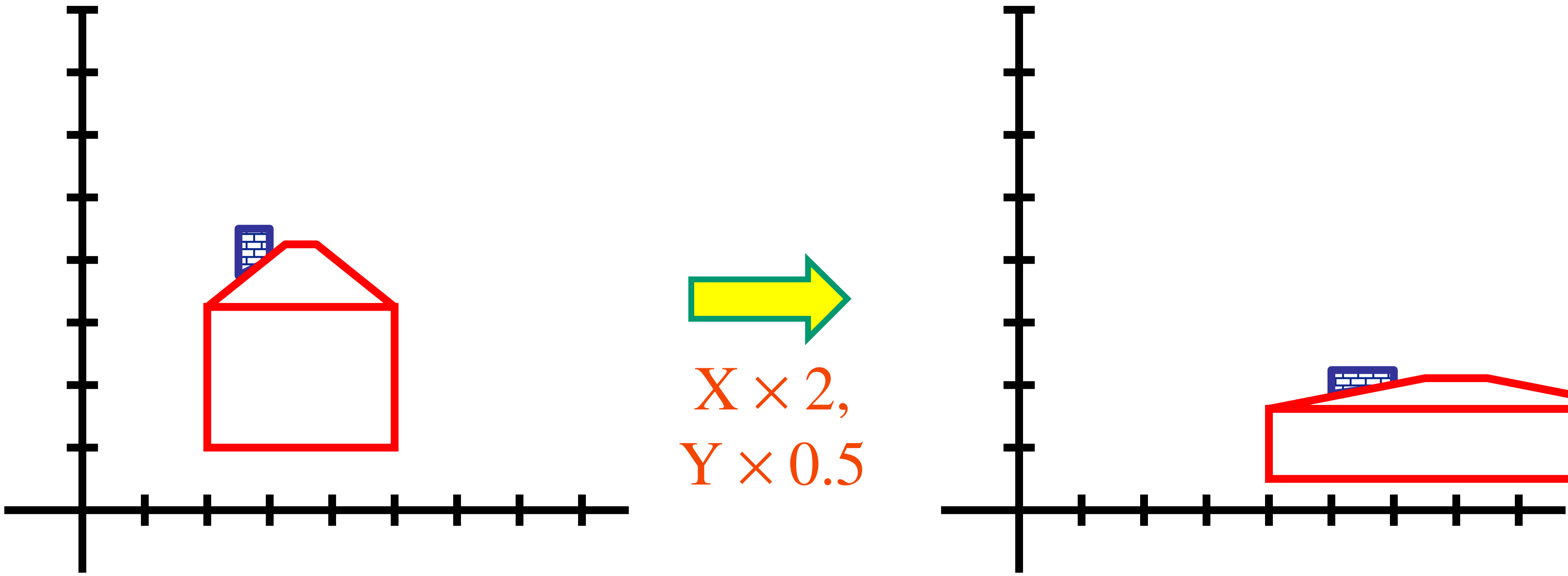
Scaling

- scaling a coordinate means multiplying each of its components by a scalar
- uniform scaling means this scalar is the same for all components:



Scaling

- non-uniform scaling: different scalars per component:



- how can we represent this in matrix form?

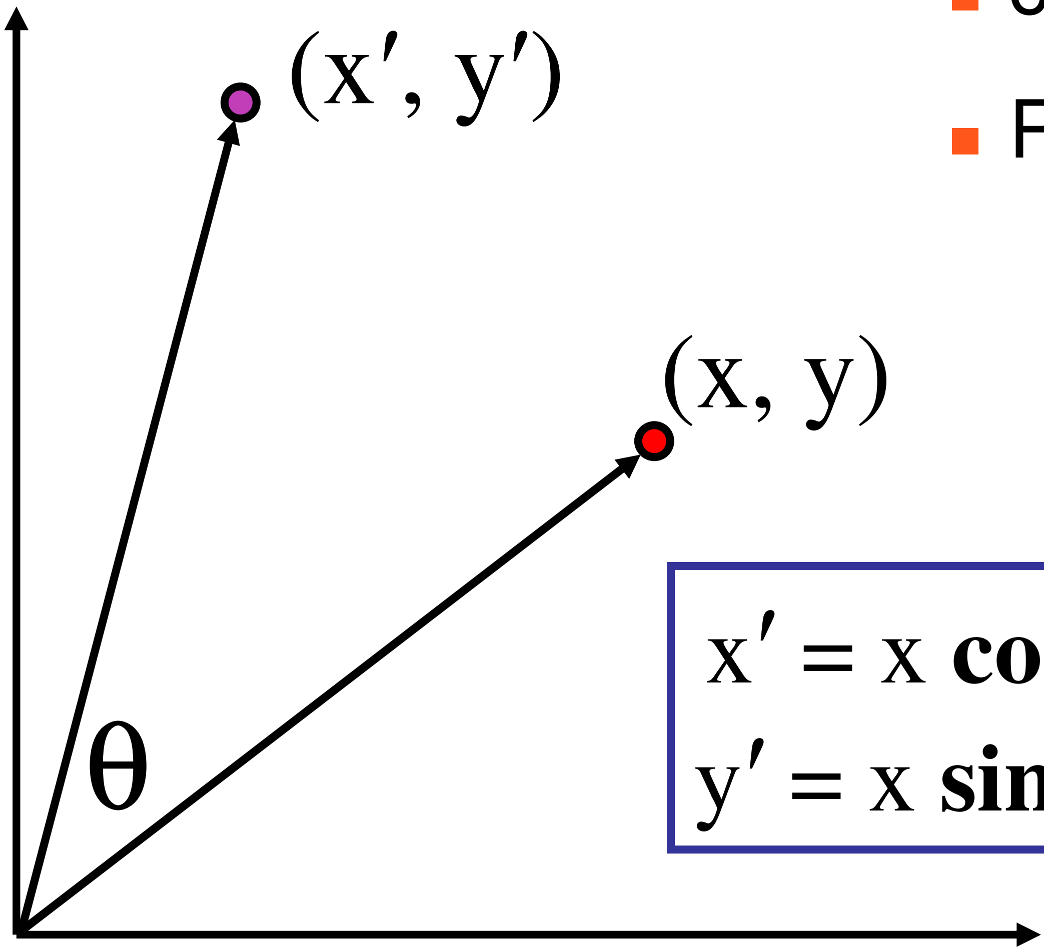
Scaling

- scaling operation: $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ax \\ by \end{bmatrix}$

- or, in matrix form: $\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\textit{scaling matrix}} \begin{bmatrix} x \\ y \end{bmatrix}$

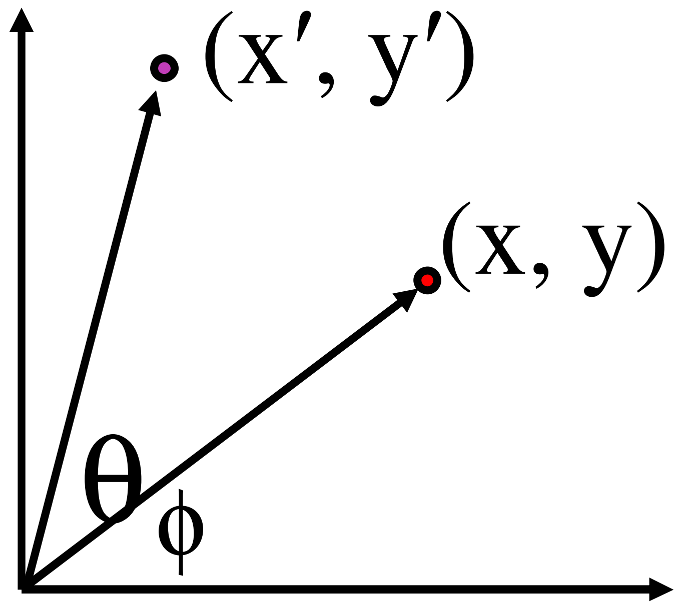
2D Rotation

- counterclockwise
- RHS



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

2D Rotation From Trig Identities



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

Trig Identity…

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

Substitute…

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

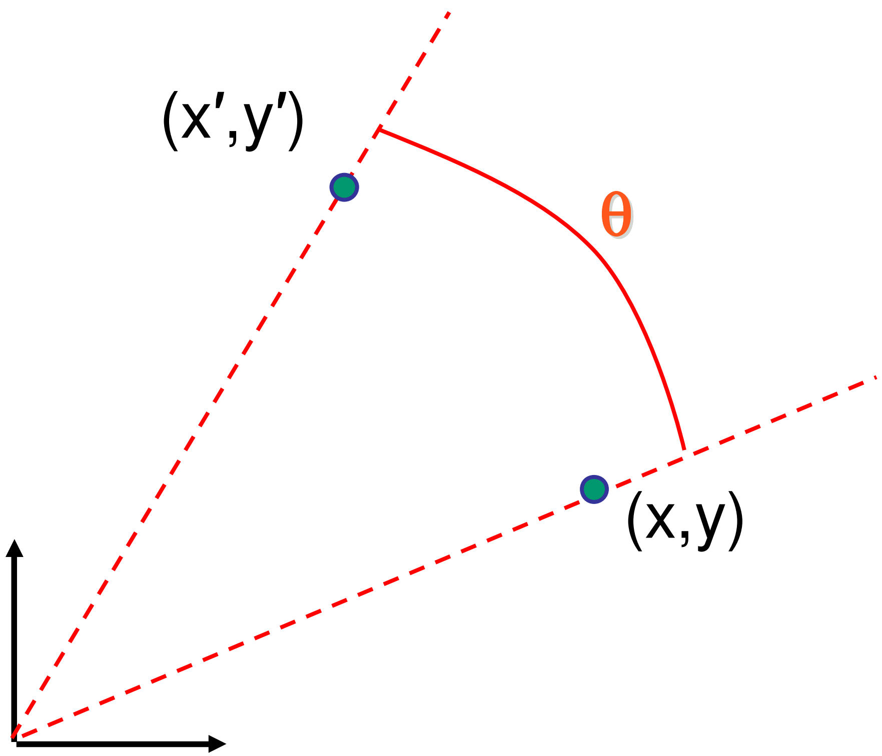
2D Rotation Matrix

- easy to capture in matrix form:

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

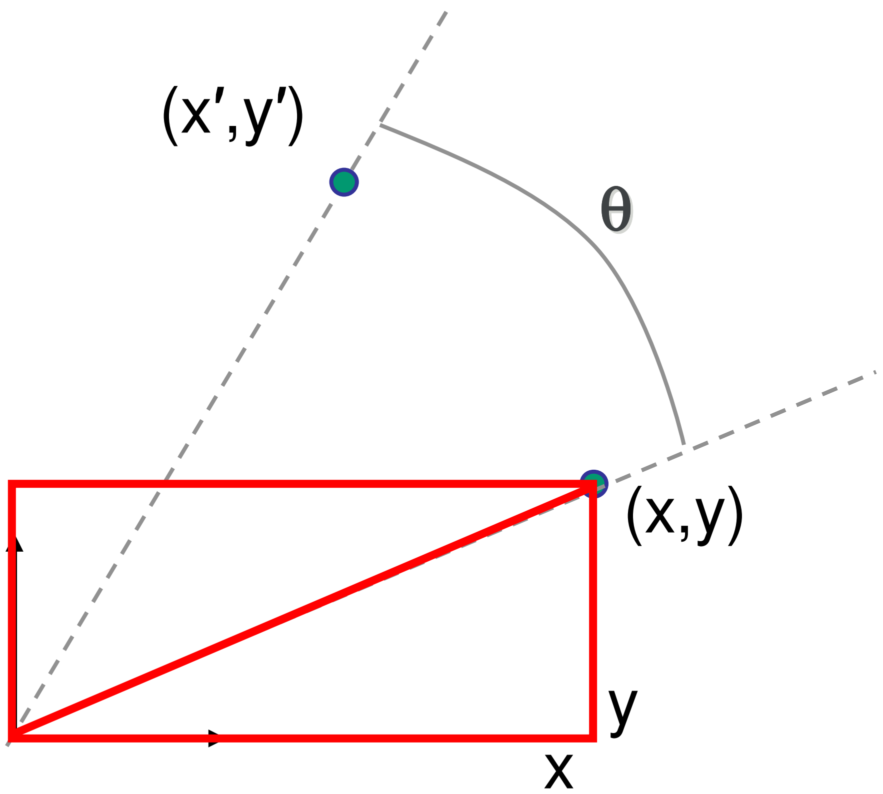
- even though sin(q) and cos(q) are nonlinear functions of q,
  - x' is a linear combination of x and y
  - y' is a linear combination of x and y

2D Rotation: Another Derivation



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

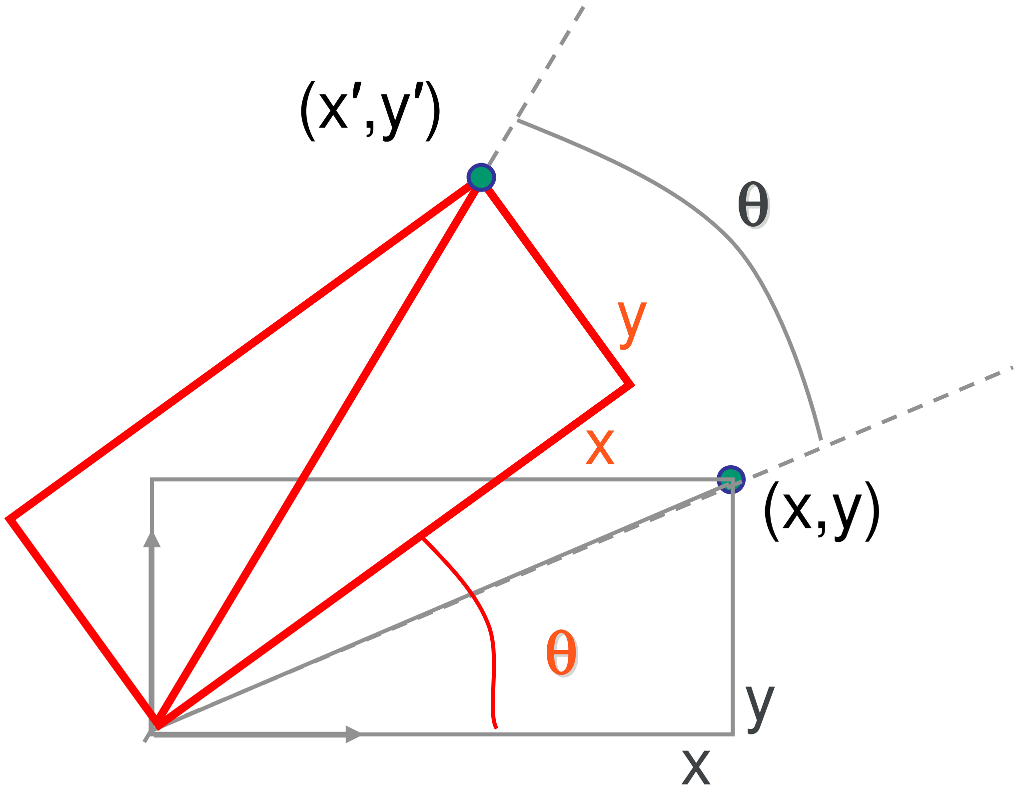
2D Rotation: Another Derivation



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

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

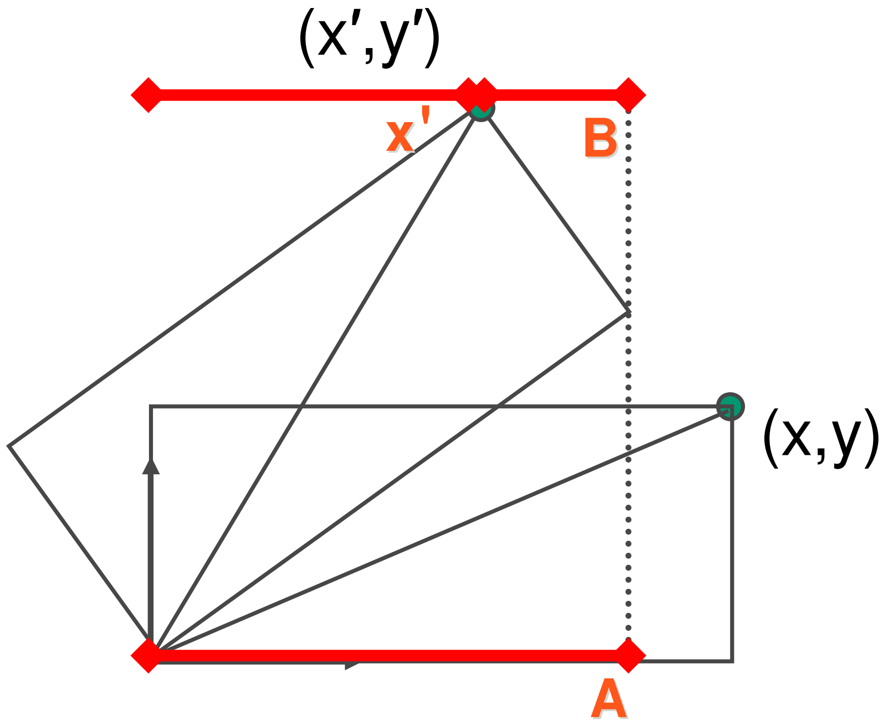
2D Rotation: Another Derivation



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

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

2D Rotation: Another Derivation

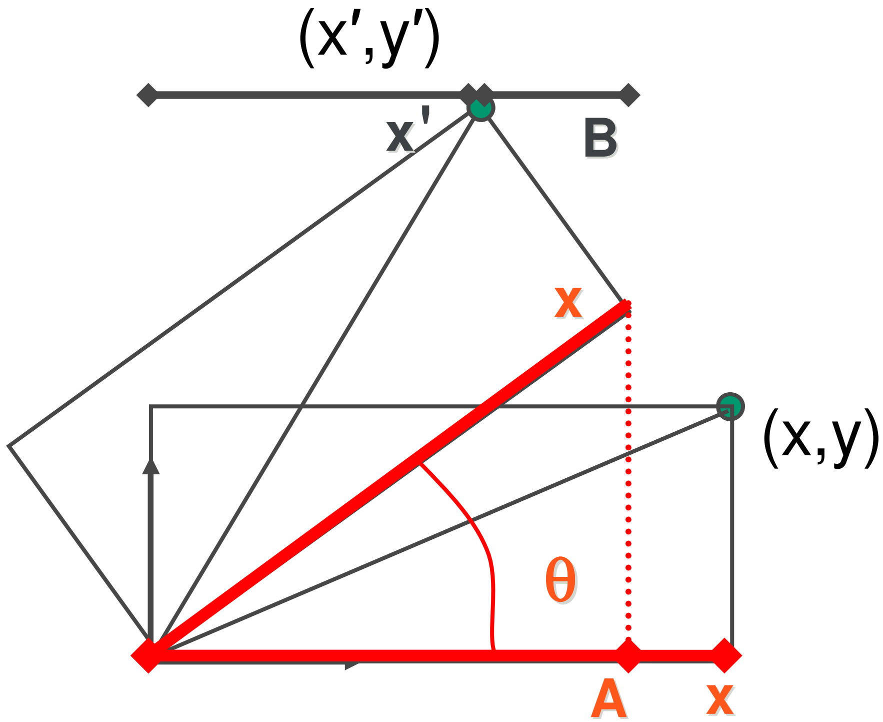


$$x' = x\cos\theta - y\sin\theta$$

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

$$x' = A - B$$

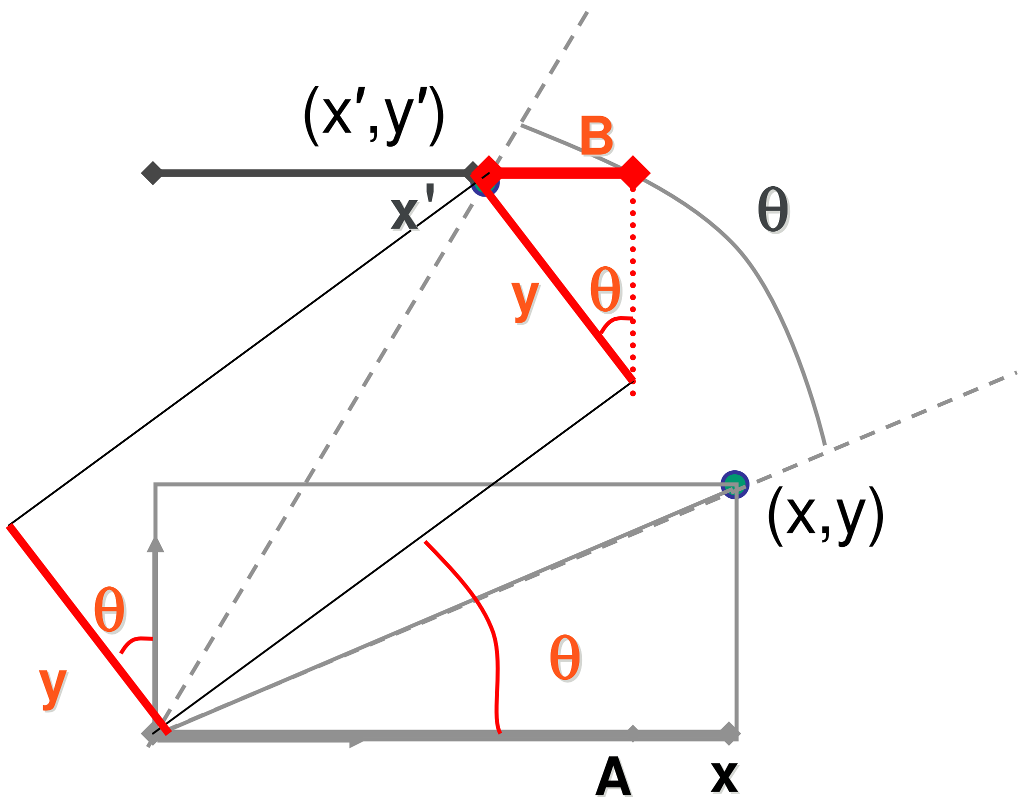
2D Rotation: Another Derivation



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

$$x' = A - B$$
$$A = x\cos\theta$$

2D Rotation: Another Derivation



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

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

$$x' = A - B$$

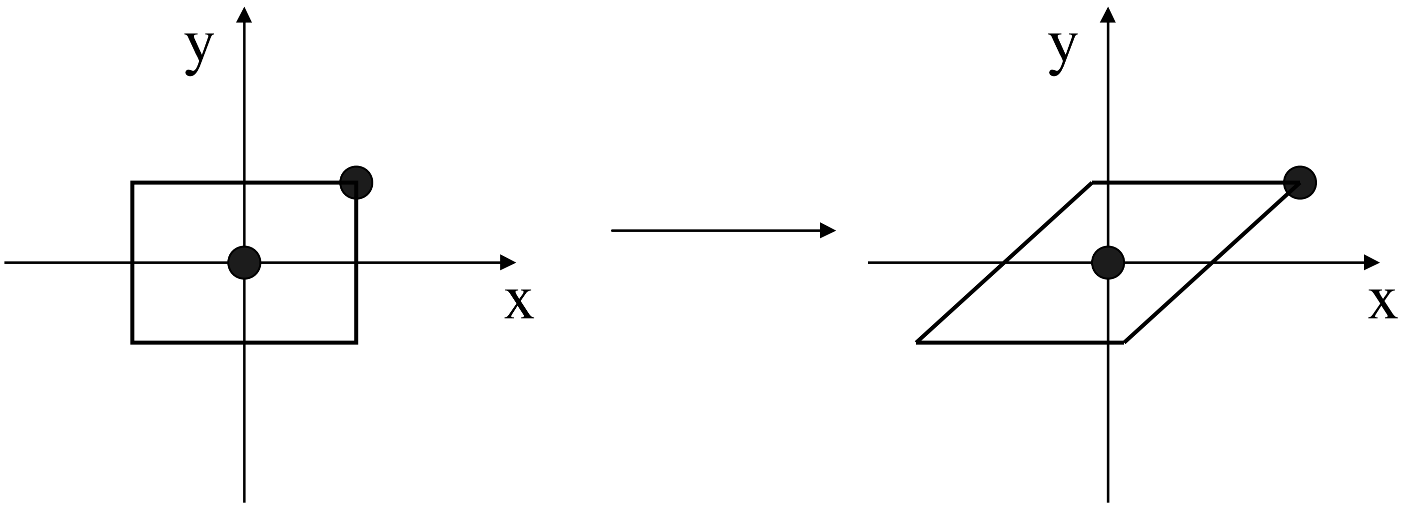
$$A = x\cos\theta$$

$$B = y\sin\theta$$

Shear

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

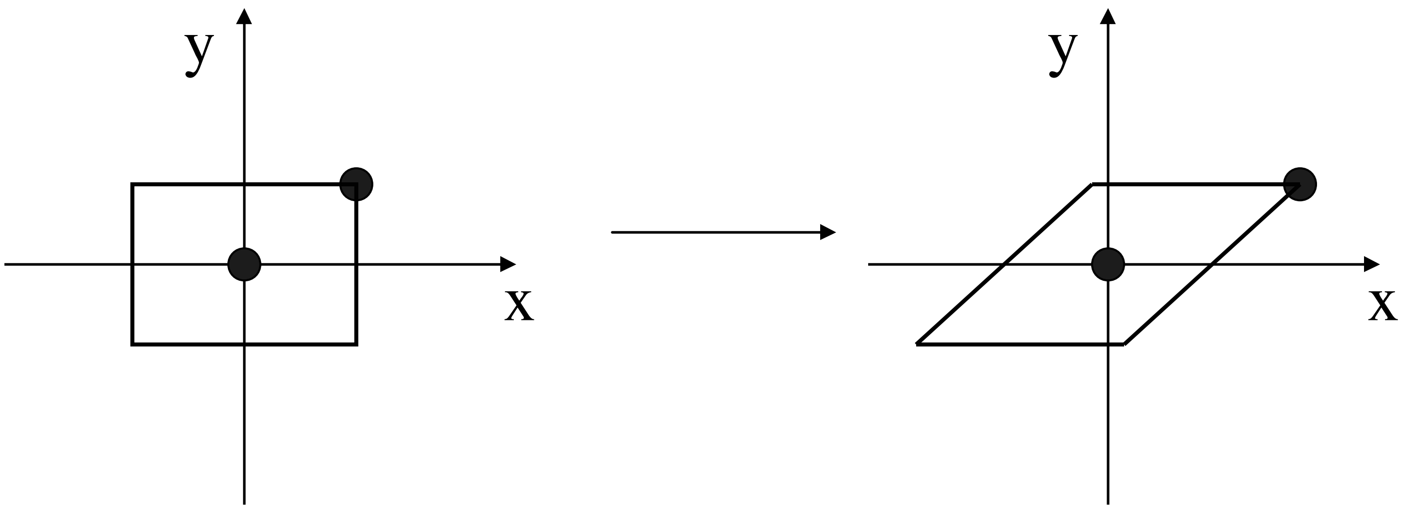
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} ? \\ ? \end{bmatrix}$$



Shear

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

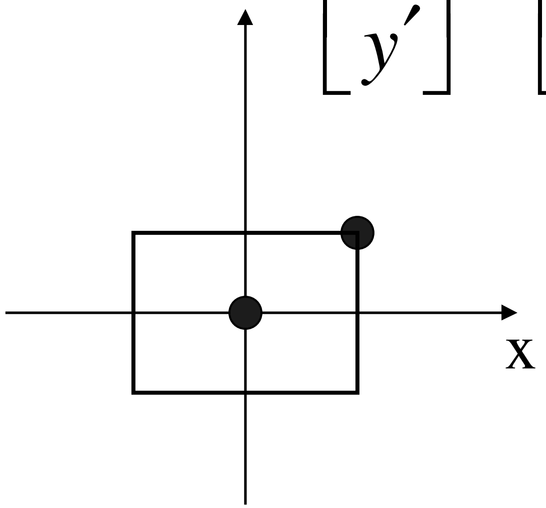
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

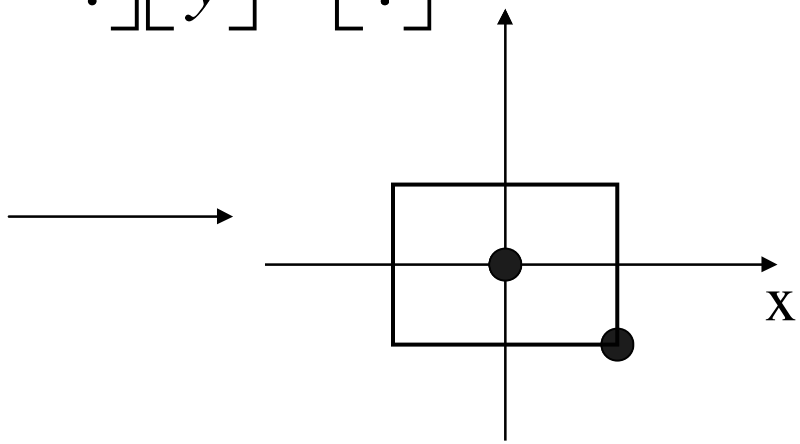


Reflection

- reflect across x axis
  - mirror

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} ? \\ ? \end{bmatrix}$$

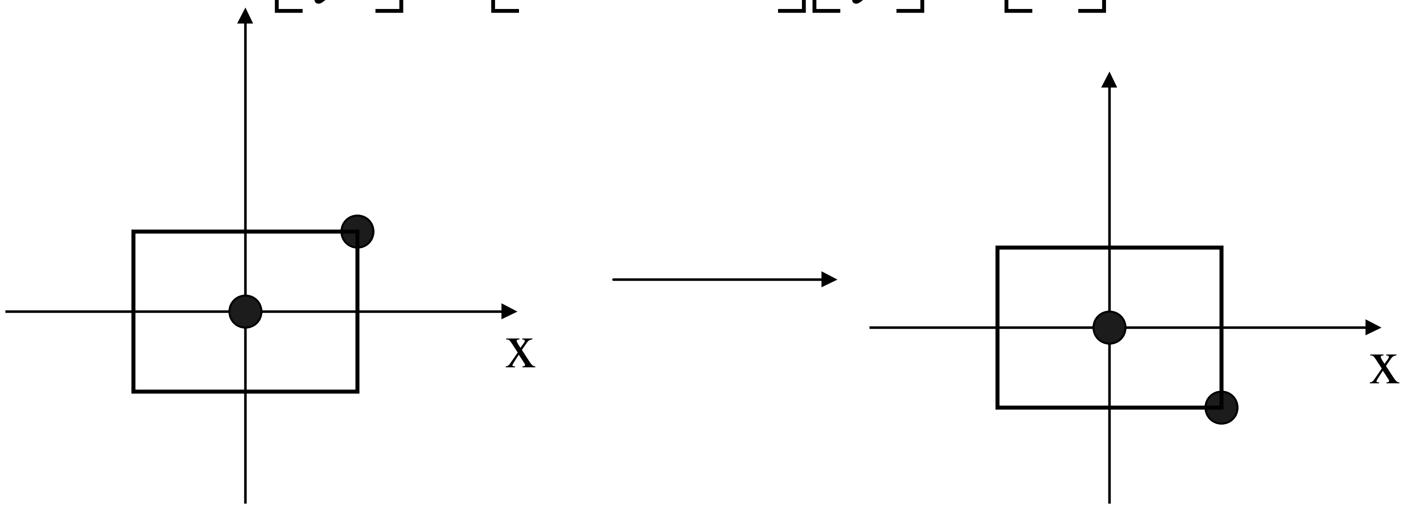




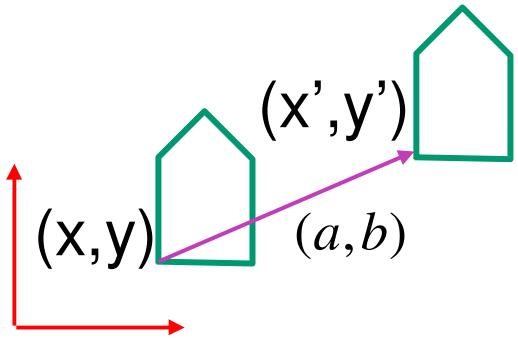
Reflection

- reflect across x axis
  - mirror

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

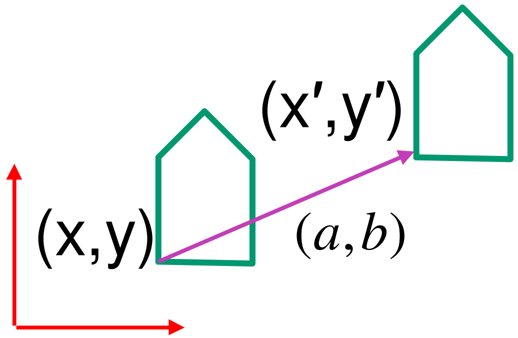


2D Translation



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

2D Translation



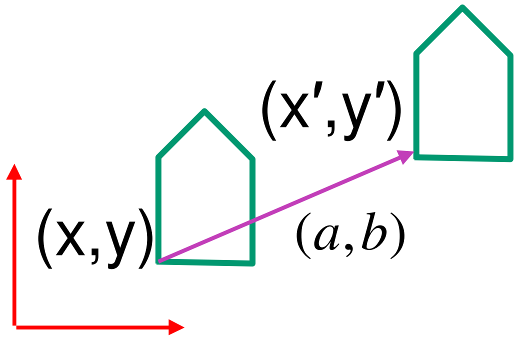
$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

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

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

scaling matrix *rotation matrix*

2D Translation



vector addition

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

matrix multiplication

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

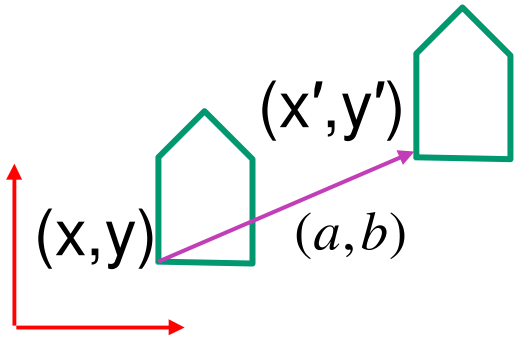
scaling matrix

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

2D Translation



vector addition

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

matrix multiplication

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

scaling matrix

matrix multiplication

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

rotation matrix

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

translation multiplication matrix??

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 $T(s\mathbf{x}+t\mathbf{y}) = s\,T(\mathbf{x}) + t\,T(\mathbf{y})$
  - origin maps to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

Challenge

- matrix multiplication
  - for everything except translation
  - how to do everything with multiplication?
    - then just do composition, no special cases
- homogeneous coordinates trick
  - represent 2D coordinates (x,y) with 3-vector (x,y,1)

Homogeneous Coordinates

- our 2D transformation matrices are now 3x3:

$$\mathbf{R}otation = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{S}cale = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

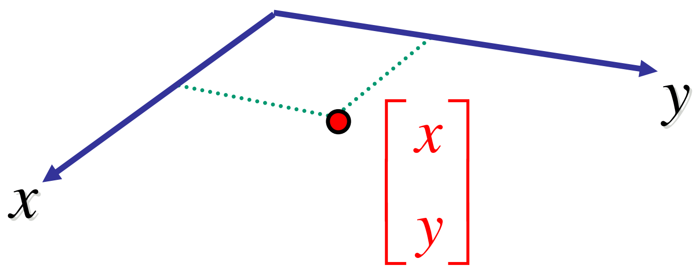
$$\mathbf{T}ranslation = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

- use rightmost column

$$\begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x*1+a*1 \\ y*1+b*1 \\ 1 \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \\ 1 \end{bmatrix}$$

Homogeneous Coordinates Geometrically

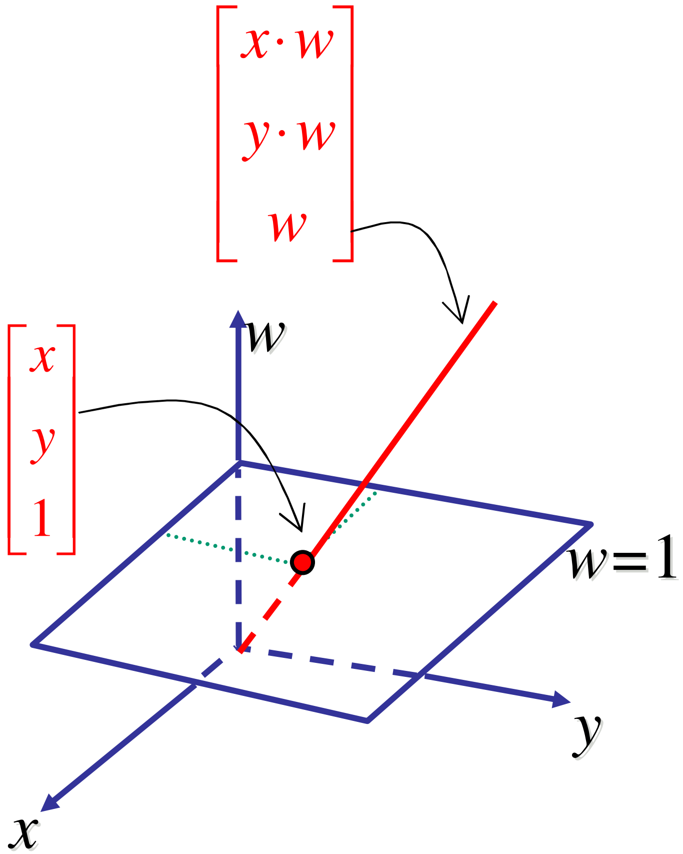
- point in 2D cartesian



Homogeneous Coordinates Geometrically

homogeneous **cartesian**

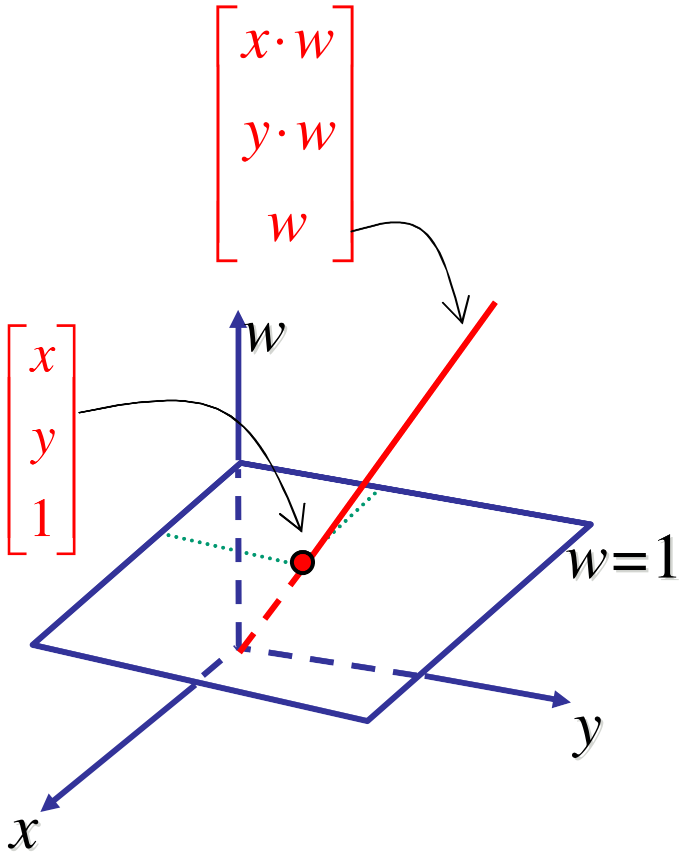
$$(x, y, w) \xrightarrow{/ w} \left(\frac{x}{w}, \frac{y}{w}\right)$$



- point in 2D cartesian + weight w = point P in 3D homog. coords
- multiples of (x,y,w)
  - form a line L in 3D
  - all homogeneous points on L represent same 2D cartesian point
  - example: (2,2,1) = (4,4,2) = (1,1,0.5)

Homogeneous Coordinates Geometrically

homogeneous		cartesian
(x, y, w)	$\xrightarrow{/\,w}$	$(\frac{x}{w}, \frac{y}{w})$



- homogenize to convert homog. 3D point to cartesian 2D point:
  - divide by w to get (x/w, y/w, 1)
  - projects line to point onto w=1 plane
- when w=0, consider it as direction
  - points at infinity
  - these points cannot be homogenized
  - lies on x-y plane
- (0,0,0) is undefined

Homogeneous Coordinates Summary

- may seem unintuitive, but they make graphics operations much easier
- allow all linear transformations to be expressed through matrix multiplication
- use 4x4 matrices for 3D transformations

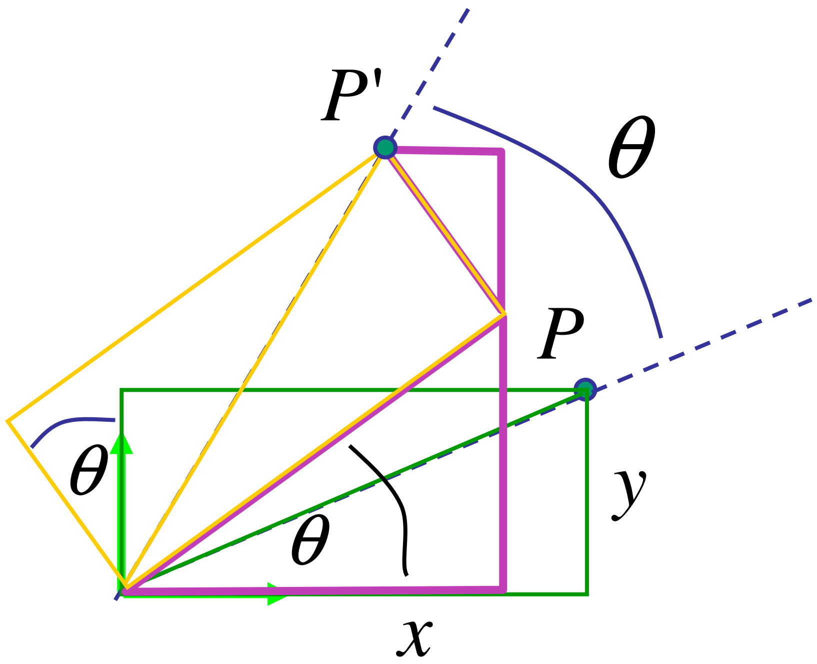
Affine Transformations

- affine transforms are combinations of
  - linear transformations
  - translations

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- properties of affine transformations
  - origin does not necessarily map 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

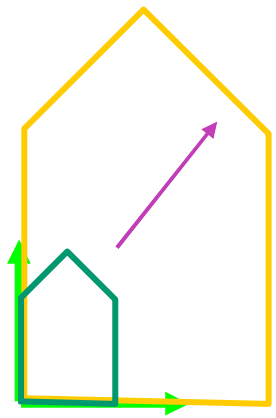
around x axis: **glRotatef(angle,1,0,0);**

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

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

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

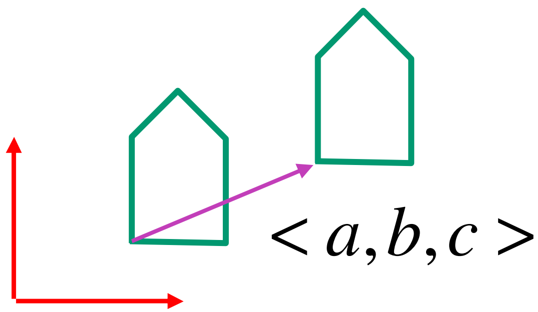
3D Scaling



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

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

- shear in x

$$xshear(sy,sz) = \begin{bmatrix} 1 & sy & sz & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- shear in y

$$yshear(sx,sz) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ sx & 1 & sz & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- shear in z

$$zshear(sx,sy) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ sx & sy & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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)

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

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

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

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

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

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

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

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

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

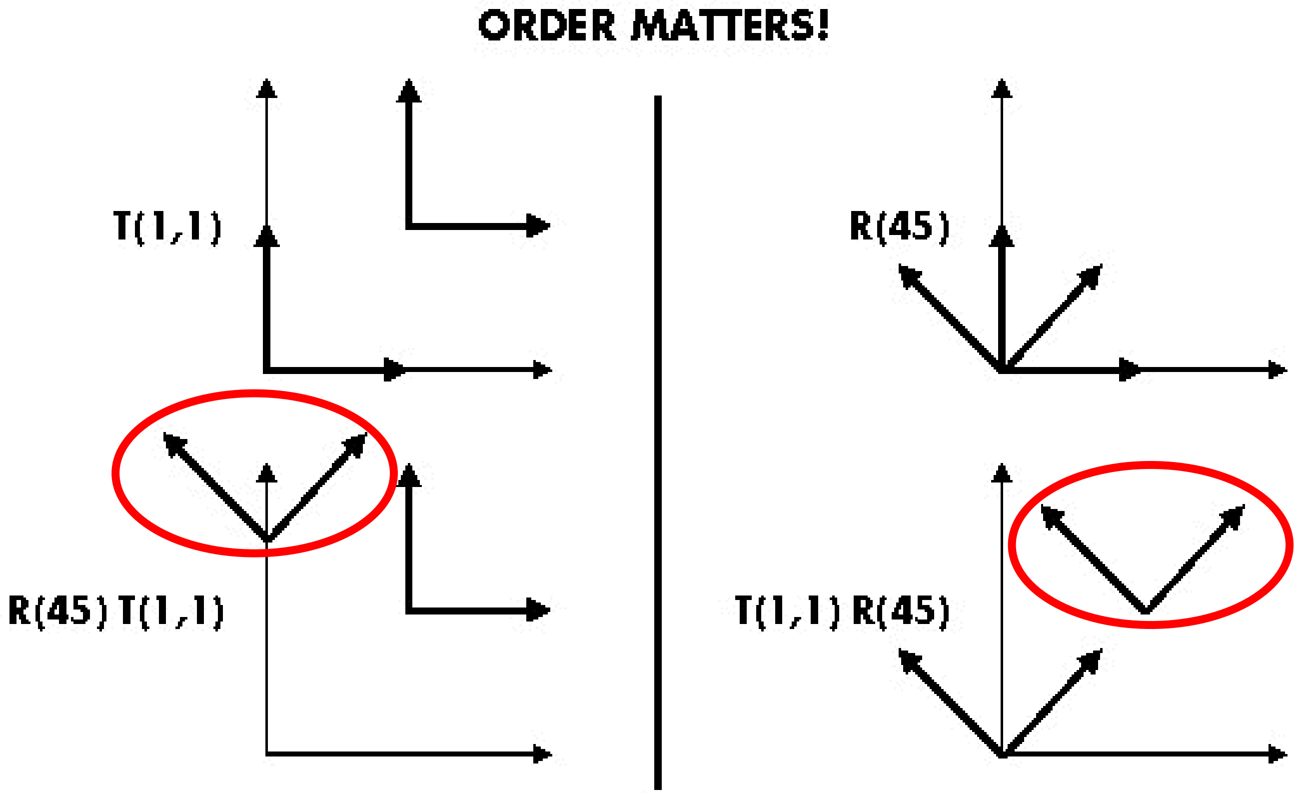
so scales multiply

- 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

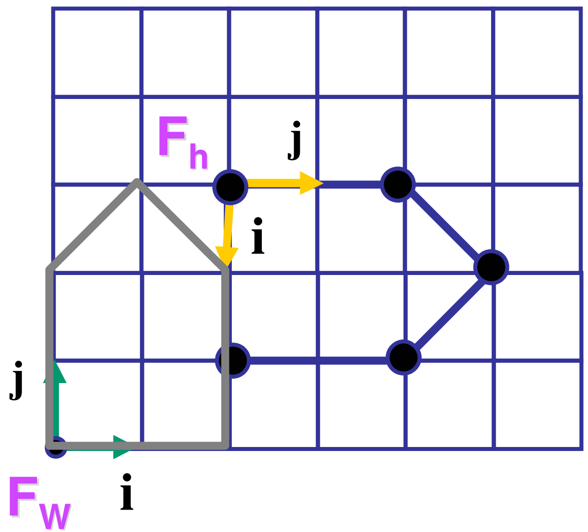
Composing Transformations



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

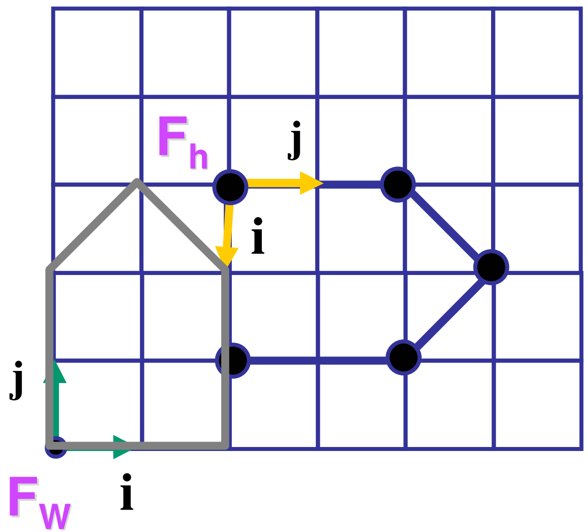
Composing Transformations

suppose we want

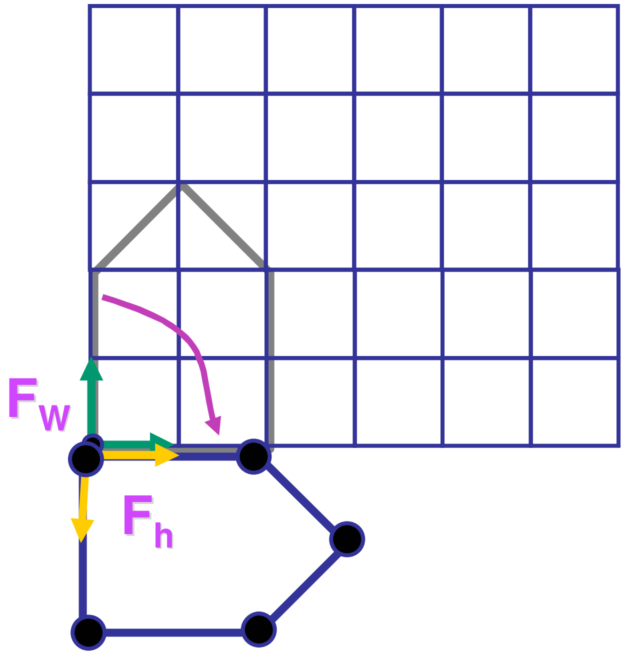


Composing Transformations

suppose we want

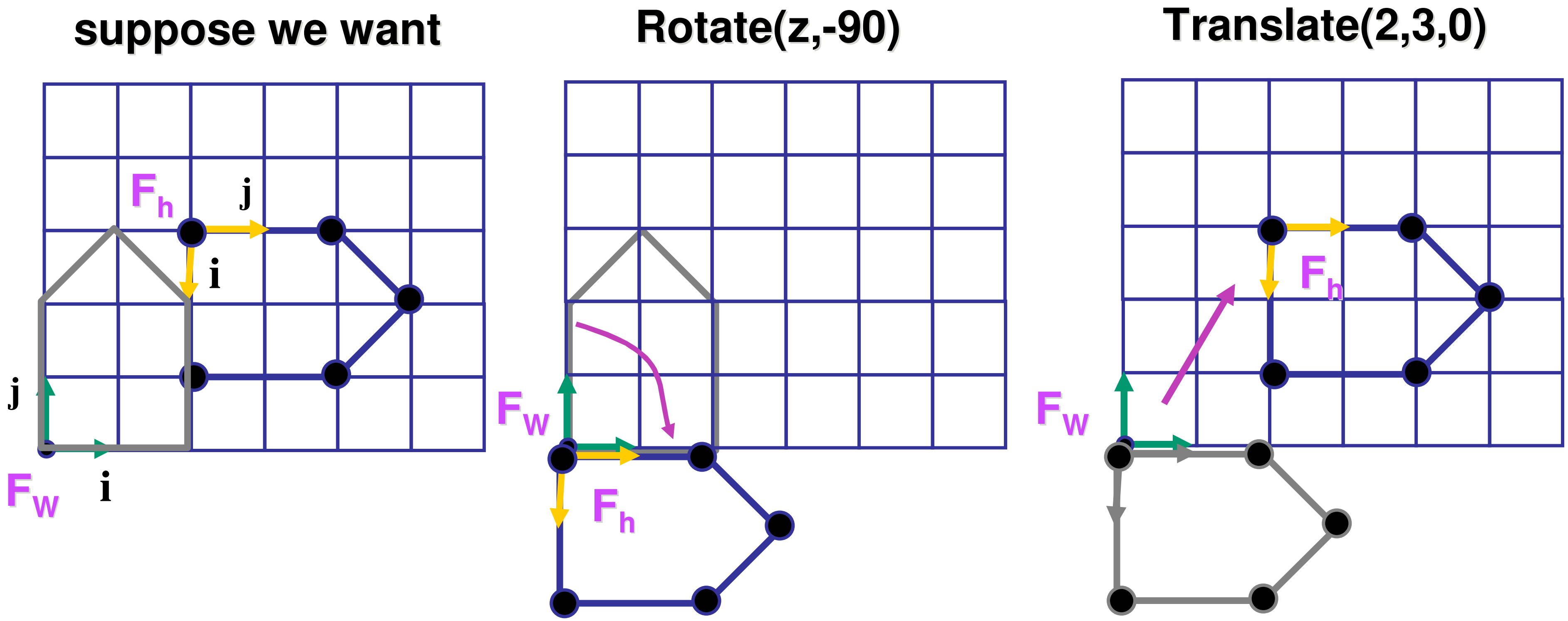


Rotate(z,-90)



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

Composing Transformations

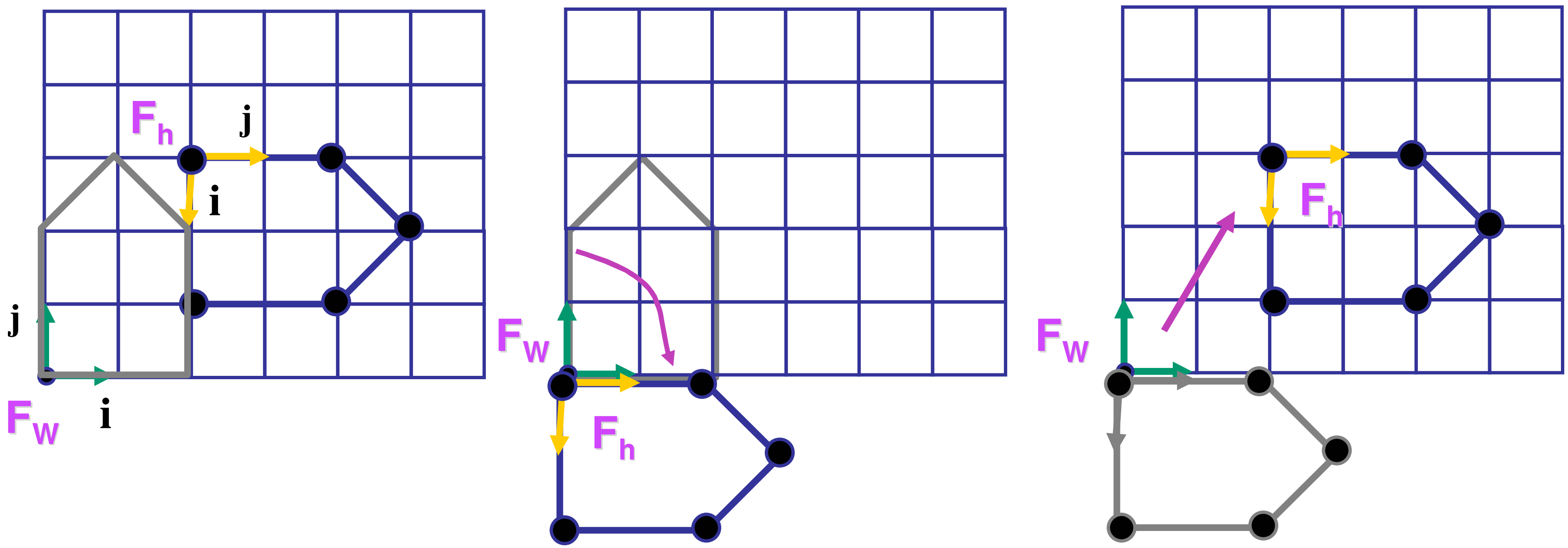


$$\mathbf{p}' = \mathbf{R}(z, -90)\,\mathbf{p}$$

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

Composing Transformations

suppose we want   Rotate(z,-90)   Translate(2,3,0)



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

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

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

Composing Transformations

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

- 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

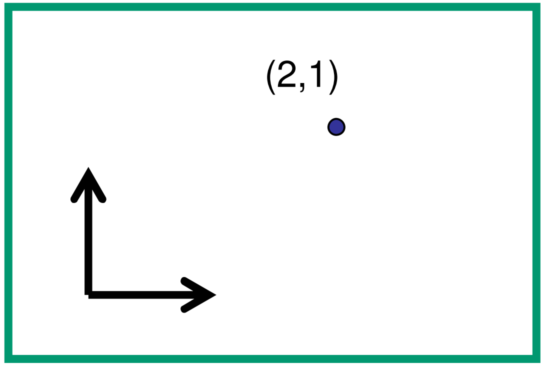
Composing Transformations

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

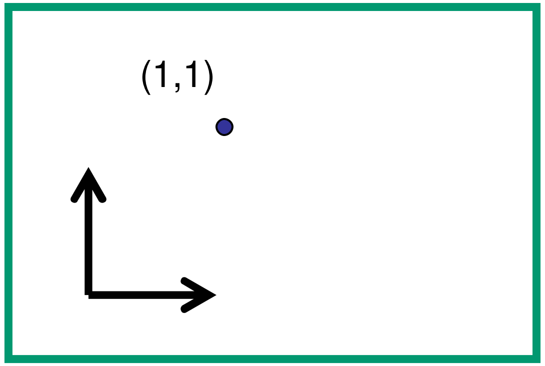
- which direction to read?
  - right to left
    - interpret operations wrt fixed coordinates
    - moving object
  - left to right **OpenGL pipeline ordering!**
    - interpret operations wrt local coordinates
    - changing coordinate system
    - OpenGL updates current matrix with postmultiply
      - 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)

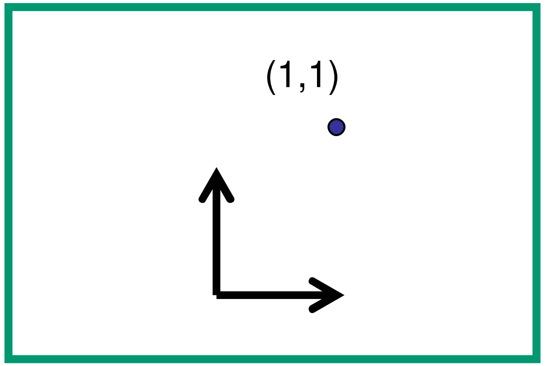


moving object



intuitive?

changing coordinate system



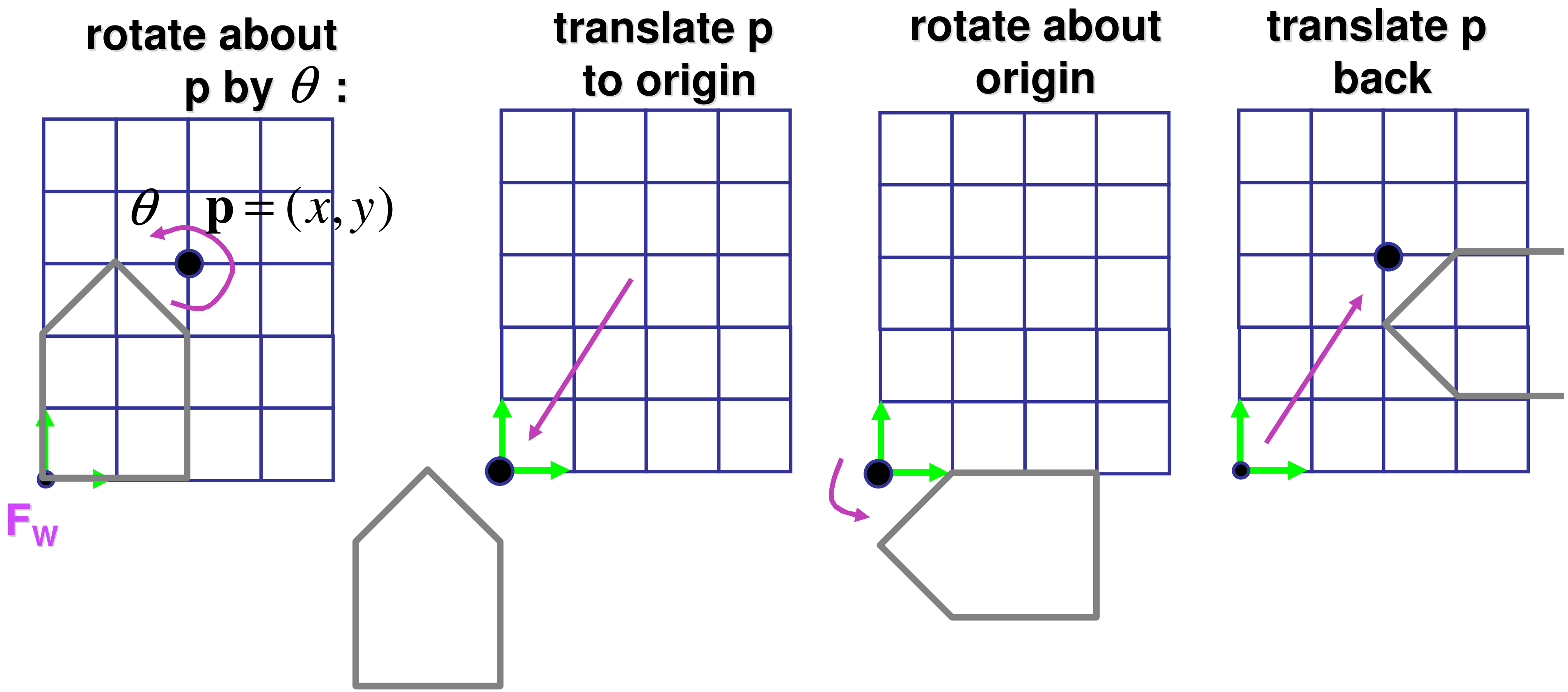
OpenGL

- 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

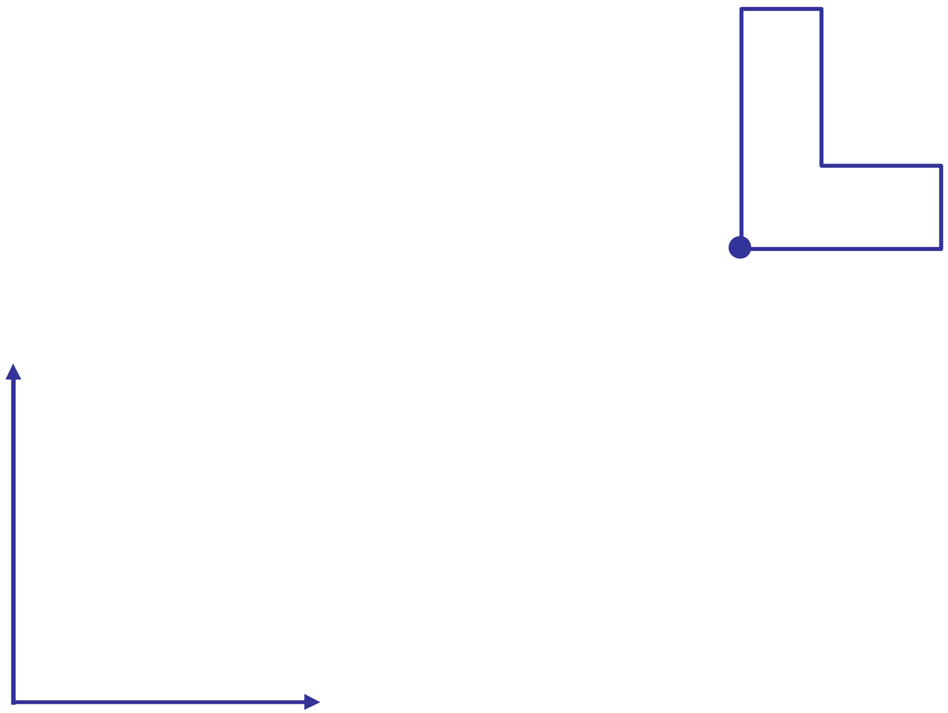
Rotation About a Point: Moving Object



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

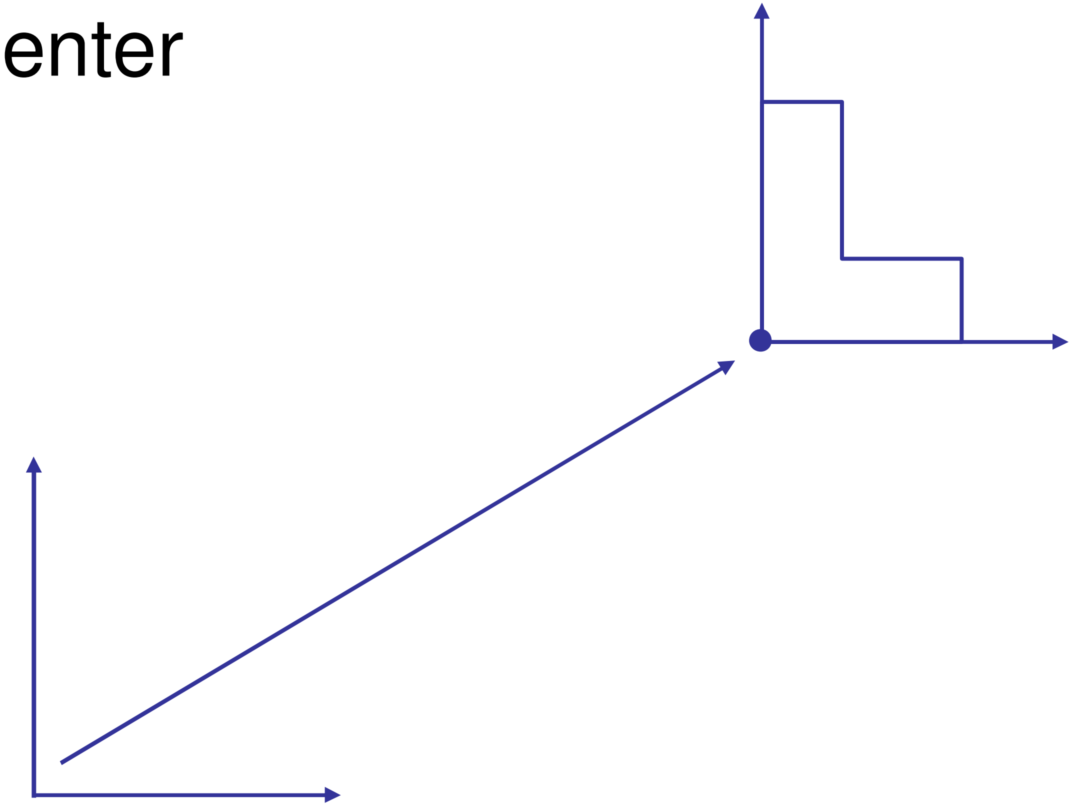
Rotation: Changing Coordinate Systems

- same example: rotation around arbitrary center



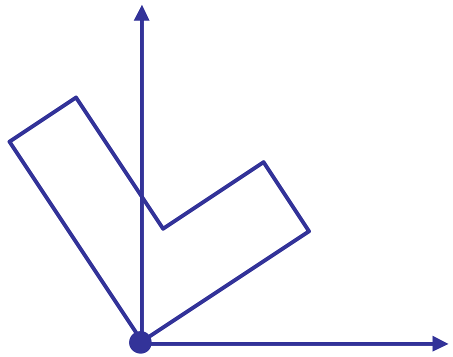
Rotation: Changing Coordinate Systems

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



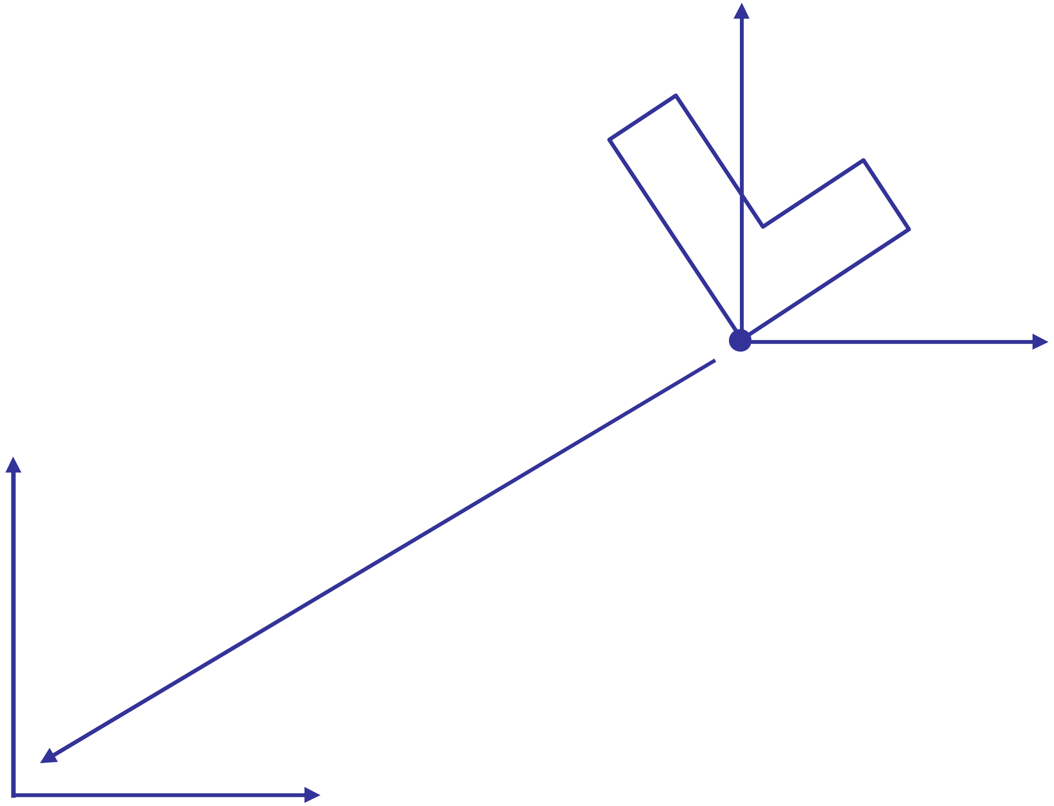
Rotation: Changing Coordinate Systems

- rotation around arbitrary center
  - step 2: perform rotation



Rotation: Changing Coordinate Systems

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



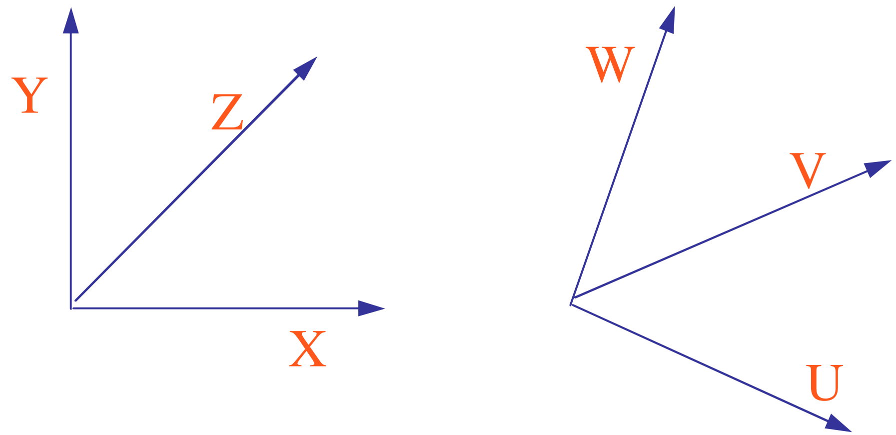
General Transform Composition

- 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



- problem:
  - given two orthonormal coordinate systems *XYZ* and *UVW*
  - find transformation from one to the other
- answer:
  - transformation matrix R whose columns are *U,V,W:*

$$R = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}$$

Arbitrary Rotation

- why?

$$R(X) = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$= (u_x, u_y, u_z)$$

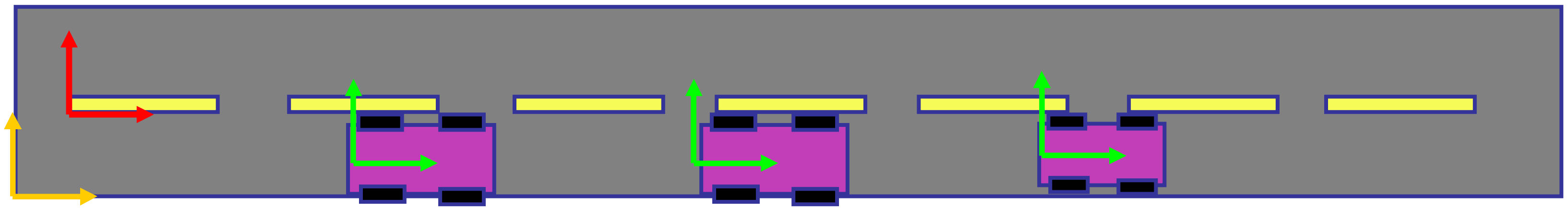
$$= U$$

- similarly $R(Y) = V$ & $R(Z) = W$

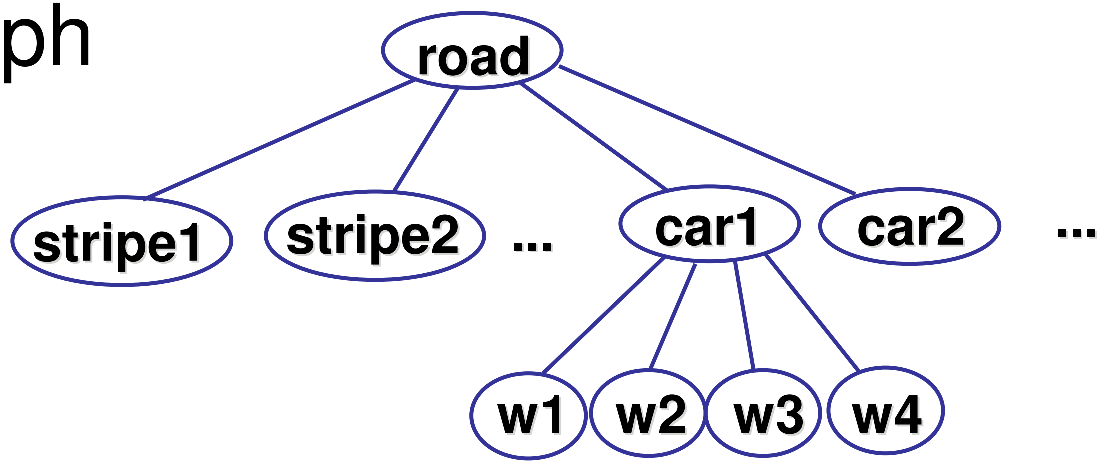
Transformation Hierarchies

Transformation Hierarchies

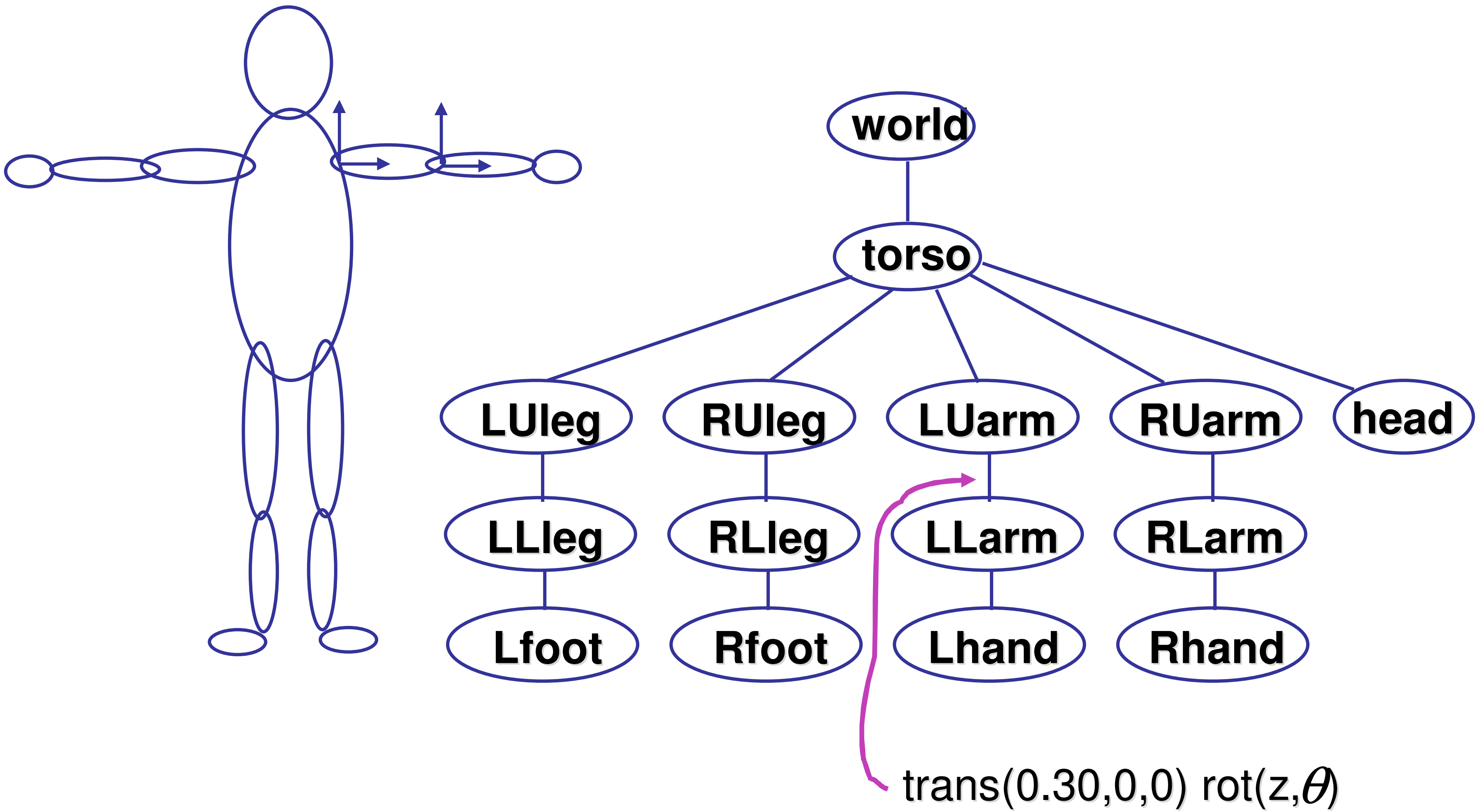
- scene may have a hierarchy of coordinate systems
  - stores matrix at each level with incremental transform from parent's coordinate system



- scene graph

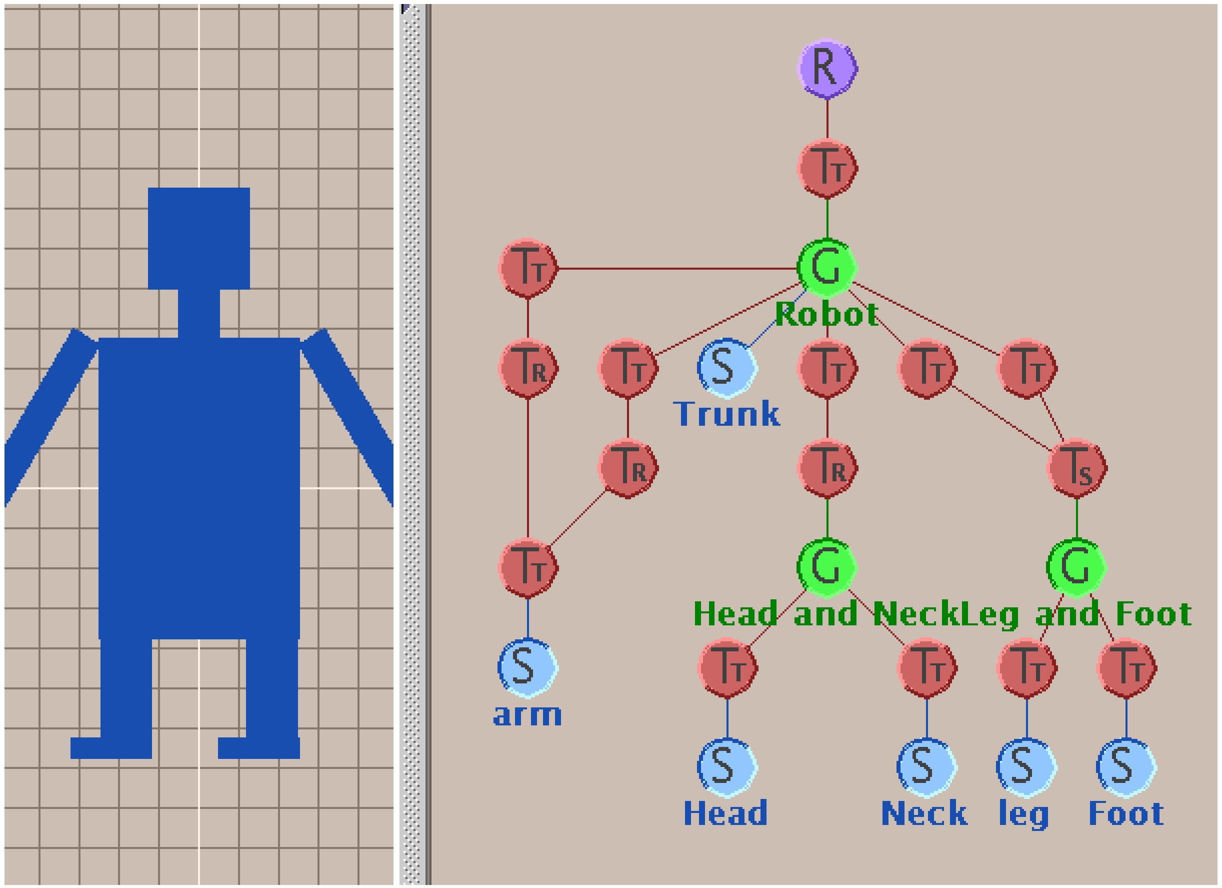


Transformation Hierarchy Example 1



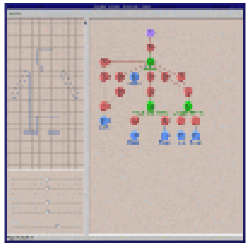
Transformation Hierarchies

- hierarchies don’t fall apart when changed
- transforms apply to graph nodes beneath



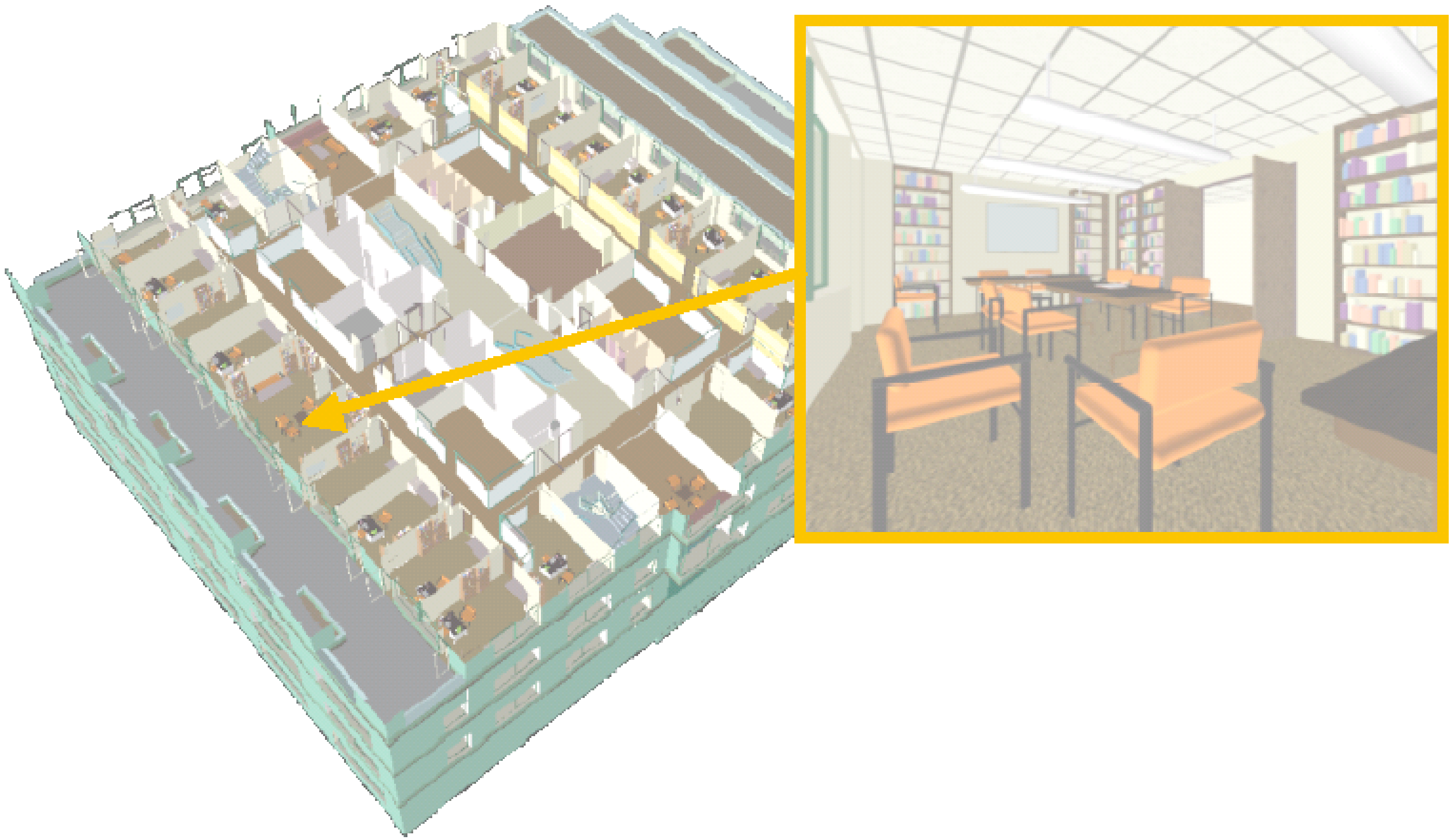
Demo: Brown Applets

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



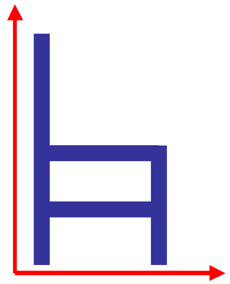
Transformation Hierarchy Example 2

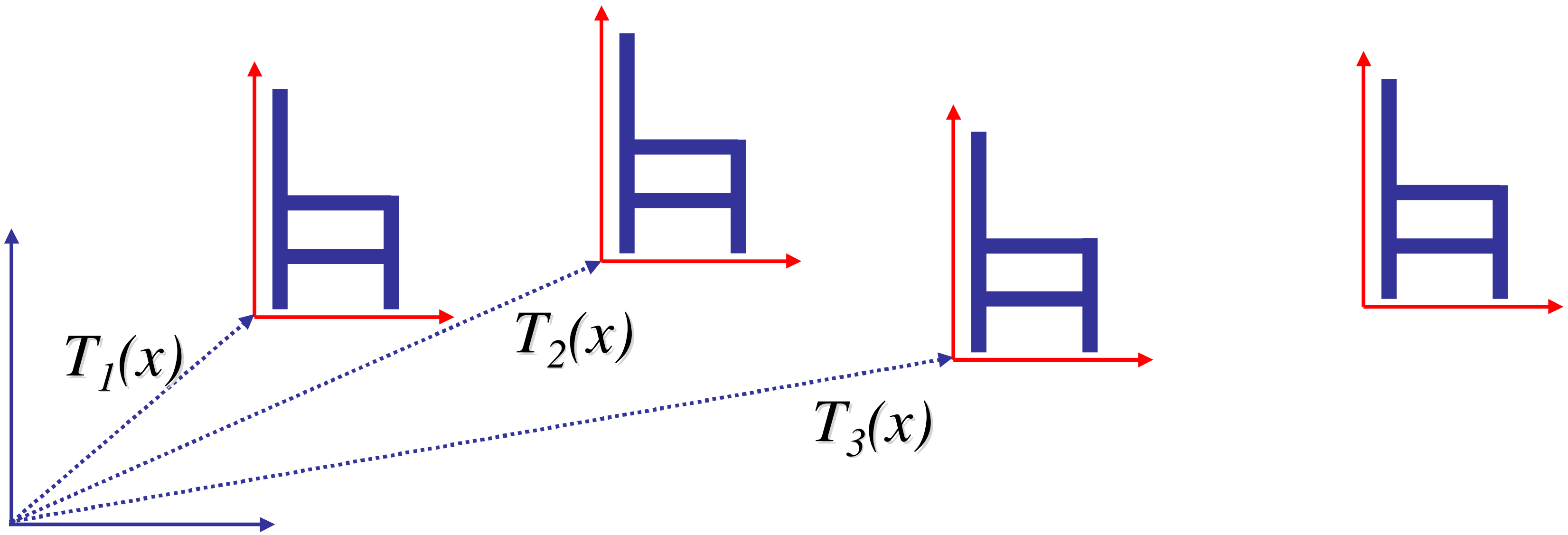
- draw same 3D data with different transformations: instancing



Matrix Stacks

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



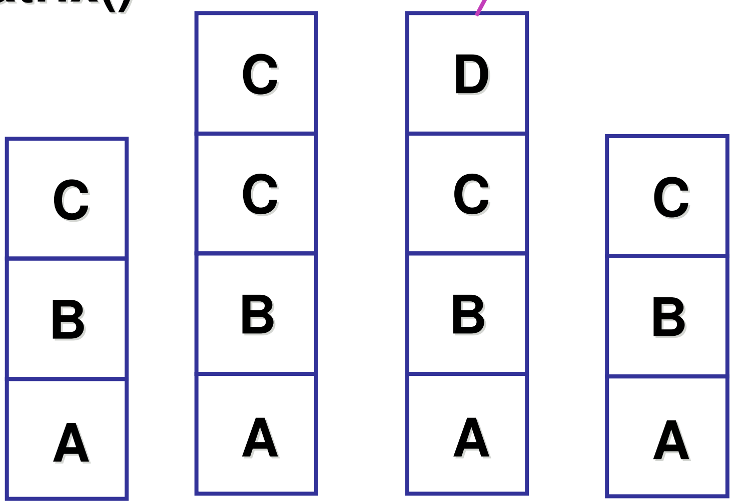


Matrix Stacks

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

glPushMatrix()

glPopMatrix()

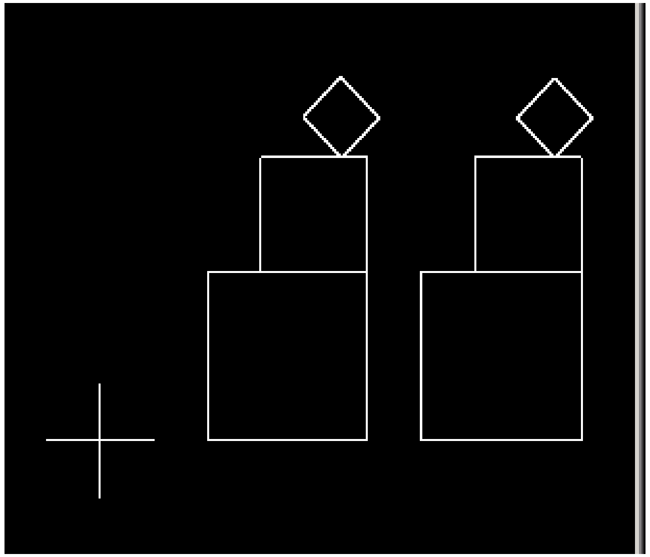


DrawSquare()
glPushMatrix()
glScale3f(2,2,2)
glTranslate3f(1,0,0)
DrawSquare()
glPopMatrix()

Modularization

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

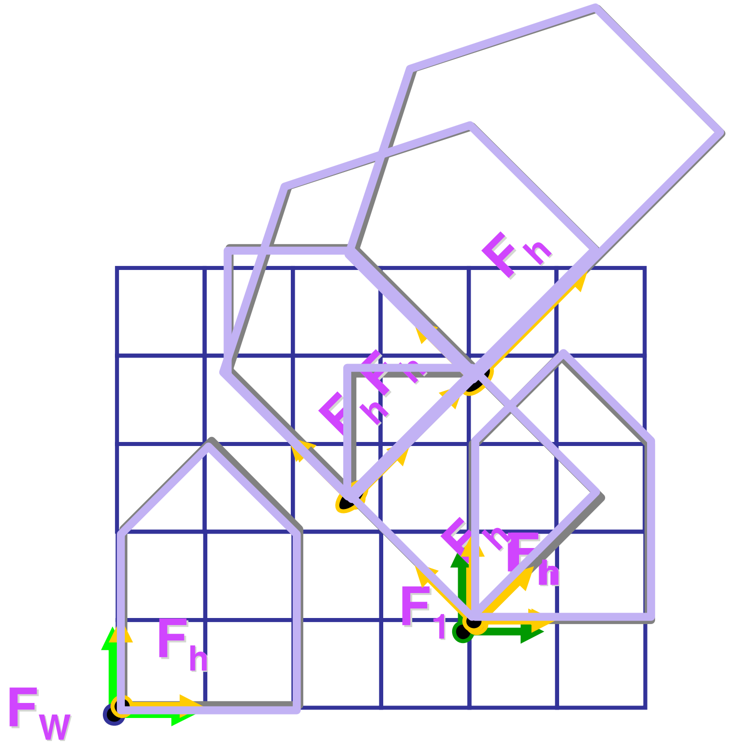
```
void drawBlock(float k) {
  glPushMatrix();

  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

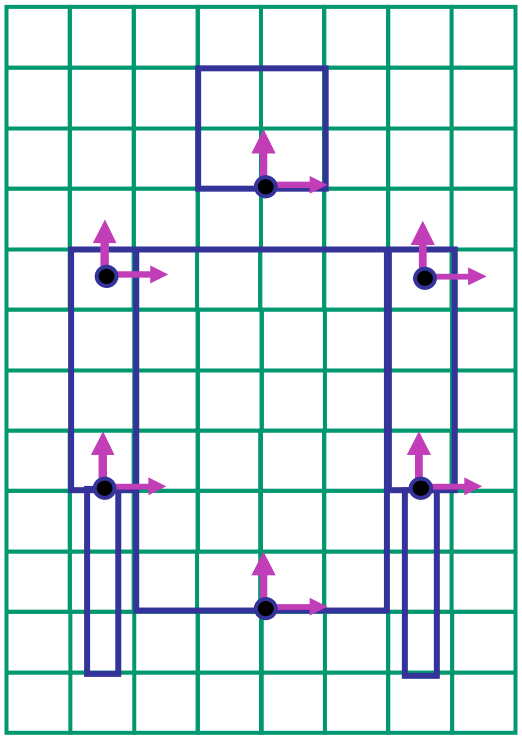
- advantages
  - no need to compute inverse matrices all the time
  - modularize changes to pipeline state
  - avoids incremental changes to coordinate systems
    - accumulation of numerical errors
- practical issues
  - in graphics hardware, depth of matrix stacks is limited
    - (typically 16 for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

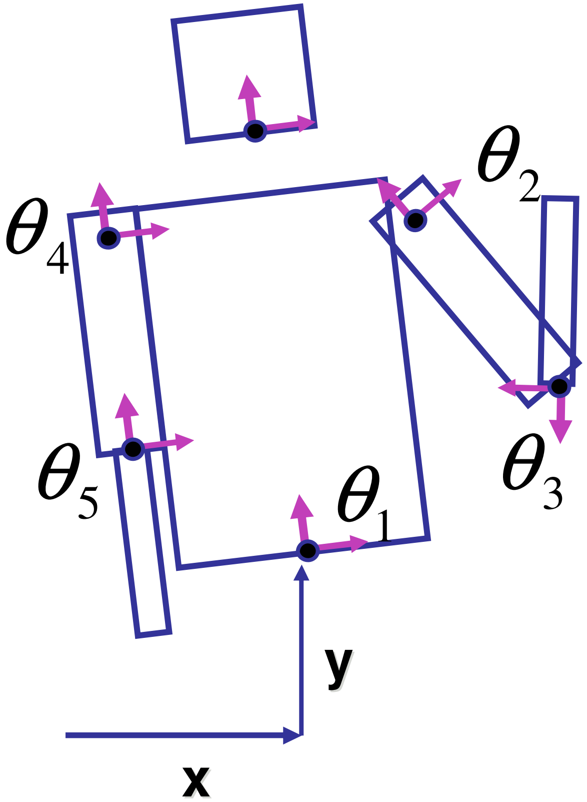
Transformation Hierarchy Example 3




```
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

Single Parameter: simple

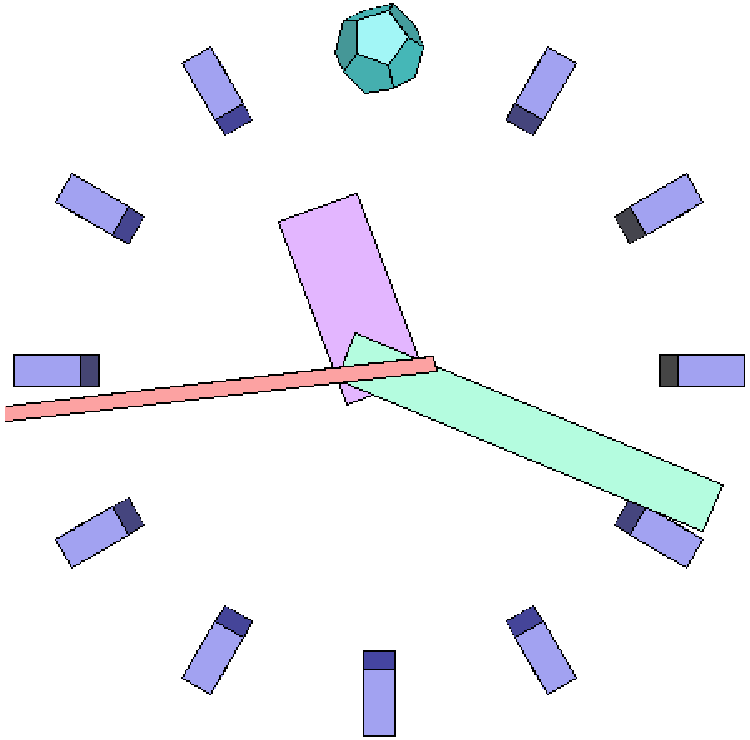
- parameters as functions of other params
  - clock: control all hands with seconds s

m = s/60, h=m/60,

theta_s = (2 pi s) / 60,

theta_m = (2 pi m) / 60,

theta_h = (2 pi h) / 60



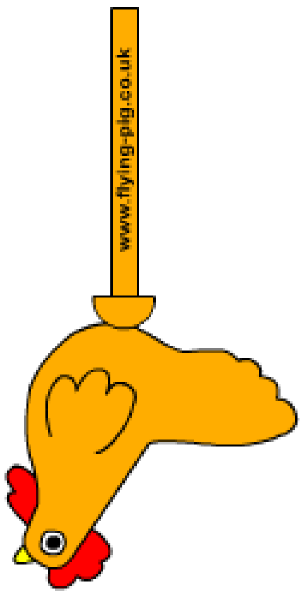
Single Parameter: complex

- mechanisms not easily expressible with affine transforms

http://www.flying-pig.co.uk

Single Parameter: complex

- mechanisms not easily expressible with affine transforms



http://www.flying-pig.co.uk/mechanisms/pages/irregular.html

Display Lists

Display Lists

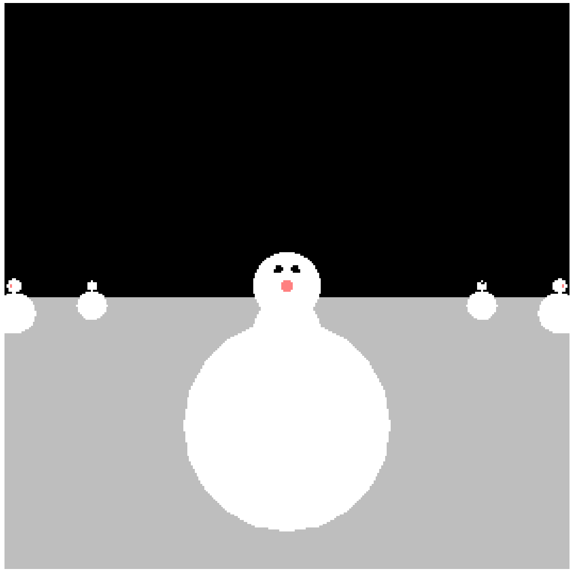
- precompile/cache block of OpenGL code for reuse
  - usually more efficient than immediate mode
    - exact optimizations depend on driver
  - good for multiple instances of same object
    - but cannot change contents, not parametrizable
  - good for static objects redrawn often
    - display lists persist across multiple frames
    - interactive graphics: objects redrawn every frame from new viewpoint from moving camera
  - can be nested hierarchically
- snowman example

  http://www.lighthouse3d.com/opengl/displaylists

One Snowman

```
void drawSnowMan() {

glColor3f(1.0f, 1.0f, 1.0f);

// Draw Body
glTranslatef(0.0f ,0.75f, 0.0f);
glutSolidSphere(0.75f,20,20);

// Draw Head
glTranslatef(0.0f, 1.0f, 0.0f);
glutSolidSphere(0.25f,20,20);

// Draw Eyes
glPushMatrix();
glColor3f(0.0f,0.0f,0.0f);
glTranslatef(0.05f, 0.10f, 0.18f);
glutSolidSphere(0.05f,10,10);
glTranslatef(-0.1f, 0.0f, 0.0f);
glutSolidSphere(0.05f,10,10);
glPopMatrix();

// Draw Nose
glColor3f(1.0f, 0.5f , 0.5f);
glRotatef(0.0f,1.0f, 0.0f, 0.0f);
glutSolidCone(0.08f,0.5f,10,2);
}
```

Instantiate Many Snowmen



```
// Draw 36 Snowmen
for(int i = -3; i < 3; i++)
  for(int j=-3; j < 3; j++) {
  glPushMatrix();
  glTranslatef(i*10.0, 0, j * 10.0);
  // Call the function to draw a snowman
  drawSnowMan();
  glPopMatrix();
}
```

36K polygons, 55 FPS

Making Display Lists

```
GLuint createDL() {
GLuint snowManDL;
// Create the id for the list
snowManDL = glGenLists(1);
glNewList(snowManDL,GL_COMPILE);
drawSnowMan();
glEndList();
return(snowManDL); }

snowmanDL = createDL();
for(int i = -3; i < 3; i++)
  for(int j=-3; j < 3; j++) {
    glPushMatrix();
    glTranslatef(i*10.0, 0, j * 10.0);
    glCallList(Dlid);
    glPopMatrix(); }
```

36K polygons, 153 FPS

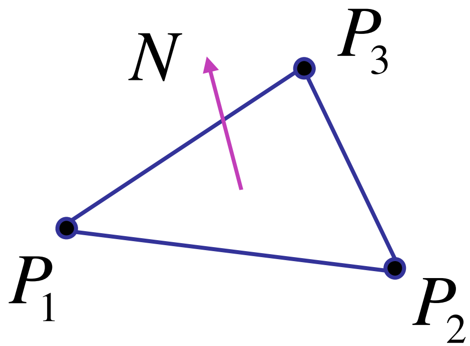
Transforming Normals

Transforming Geometric Objects

- lines, polygons made up of vertices
- just transform the vertices, interpolate between
- does this work for everything? no!

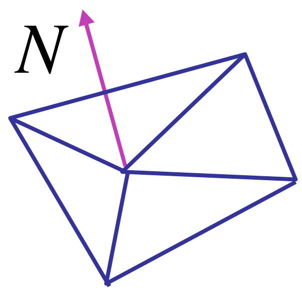
Computing Normals

- polygon:



$$N = (P_2 - P_1) \times (P_3 - P_1)$$

- assume vertices ordered CCW when viewed from visible side of polygon
- normal for a vertex
  - specify polygon orientation
  - used for lighting
  - supplied by model (i.e., sphere), or computed from neighboring polygons



Transforming Normals

- what is a normal?
  - a **direction**
    - homogeneous coordinates: w=0 means direction
  - often normalized to unit length
  - vs. points/vectors that are object vertex locations
- what are normals for?
    - specify orientation of polygonal face
    - used when computing lighting

$$\begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} Nx' \\ Ny' \\ Nz' \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & T_x \\ m_{21} & m_{22} & m_{23} & T_y \\ m_{31} & m_{32} & m_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Nx \\ Ny \\ Nz \\ 0 \end{bmatrix}$$

- so if points transformed by matrix **M**, can we just transform normal vector by **M** too?

Transforming Normals

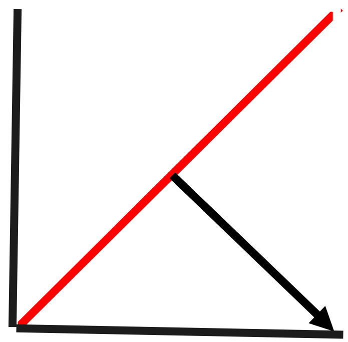
$$\begin{bmatrix} x' \\ y' \\ z' \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & T_x \\ m_{21} & m_{22} & m_{23} & T_y \\ m_{31} & m_{32} & m_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

- translations OK: w=0 means unaffected
- rotations OK
- uniform scaling OK

- these all maintain direction

Transforming Normals

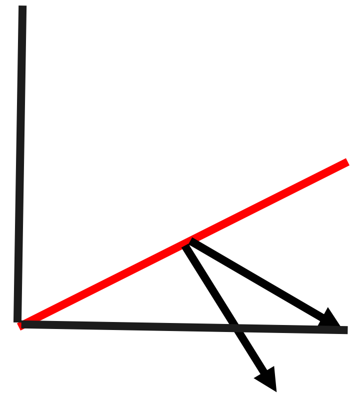
- nonuniform scaling does not work
- x-y=0 plane
  - line x=y
  - normal: [1,-1,0]
    - direction of line x=-y
    - (ignore normalization for now)



Transforming Normals

- apply nonuniform scale: stretch along x by 2
  - new plane x = 2y
- transformed normal: [2,-1,0]

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



  - normal is direction of line x = -2y or x+2y=0
  - not perpendicular to plane!
  - should be direction of 2x = -y

Planes and Normals

- plane is all points perpendicular to normal
  - $N \cdot P = 0$ (with dot product)
  - $N^{\mathbf{T}} P = 0$ (matrix multiply requires transpose)

$$N = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}, P = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- explicit form: plane = $ax + by + cz + d$

Finding Correct Normal Transform

- transform a plane

$$P \qquad\longrightarrow\qquad P' = MP$$
$$N \qquad\longrightarrow\qquad N' = QN$$

given M,
what should Q be?

$$N'^T P' = 0$$

stay perpendicular

$$(QN)^T (MP) = 0$$

substitute from above

$$N^T Q^T M P = 0$$

$$(\mathbf{AB})^\mathbf{T} = \mathbf{B}^\mathbf{T}\mathbf{A}^\mathbf{T}$$

$$Q^T M = I$$

$$\mathbf{N}^\mathbf{T}\mathbf{P} = 0 \text{ if } \mathbf{Q}^\mathbf{T}\mathbf{M} = \mathbf{I}$$

$$\mathbf{Q} = \left(\mathbf{M}^{-1}\right)^\mathbf{T}$$

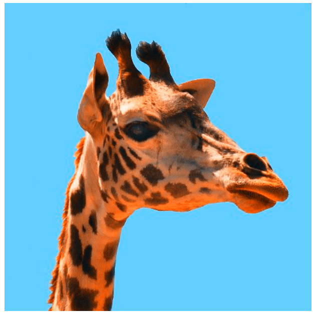
thus the normal to any surface can be transformed by the inverse transpose of the modelling transformation

Assignments

Assignments

- project 1
  - out today, due 11:59pm Wed May 18
    - you should start very soon!
  - build giraffe out of cubes and 4x4 matrices
    - think cartoon, not beauty
  - template code gives you program shell, Makefile
    - http://www.ugrad.cs.ubc.ca/~cs314/Vmay2005/p1.tar.gz
- written homework 1
  - out today, due 4pm Wed May 18
  - theoretical side of material

Real Giraffes

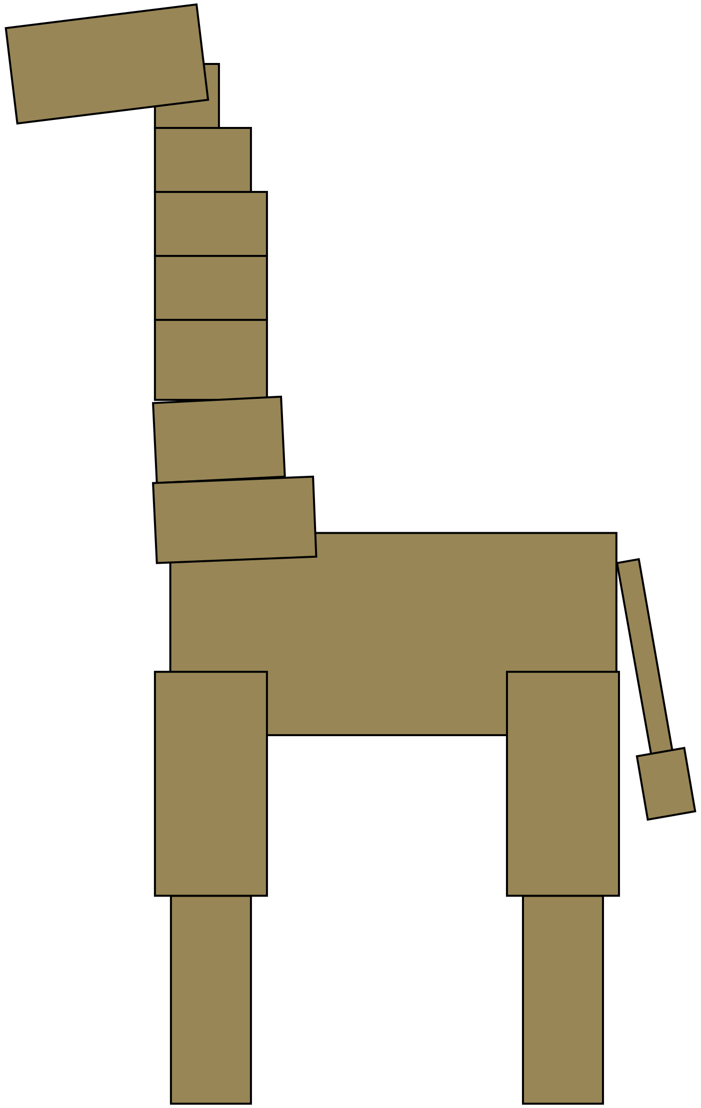


www.buffaloworks.us/images/giraffe.jpg

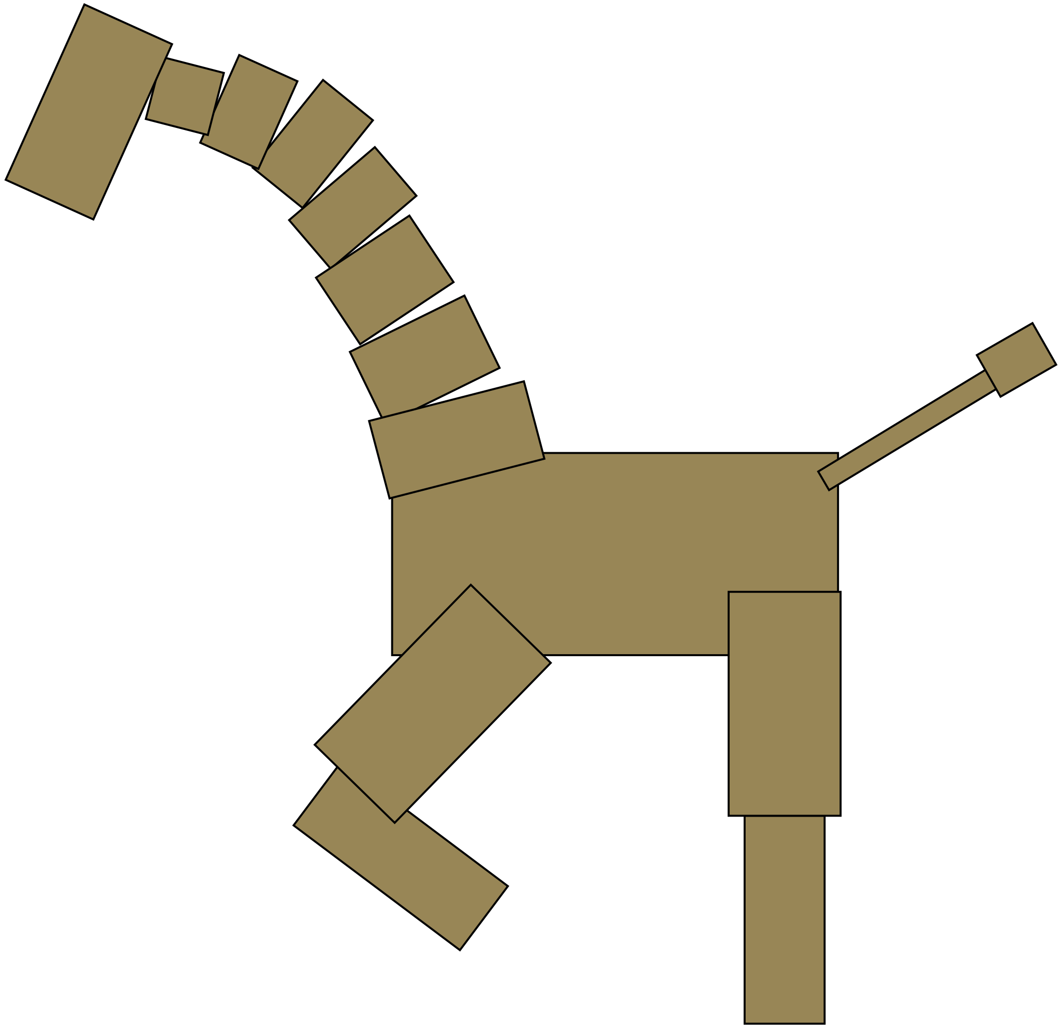
cgi.di.uoa.gr/~kmorfo/Images/Toronto/Giraffe.jpg

www.giraffes.org/graffe.jpg

Articulated Giraffe



Articulated Giraffe



Demo

Project 1 Advice

- build then animate one section at a time
  - ensure you’re constructing hierarchy correctly
  - use body as scene graph root
  - start with an upper leg
- consider using separate transforms for animation and modelling
- make sure you redraw exactly and only when necessary

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
- thinking about transformations
  - move physical objects around
  - play with demos
    - Brown scenegraph applets

Project 1 Advice

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

Labs Reminder

- in CICSR 011
- today 3-4, 4-5
  - Thu labs are for help with programming projects
    - Thursday 11-12 slot deprecated first four weeks
  - Tue labs are for help with written assignments
    - Tuesday 11-12 slot is fine
  - no separate materials to be handed in
- after-hours door code