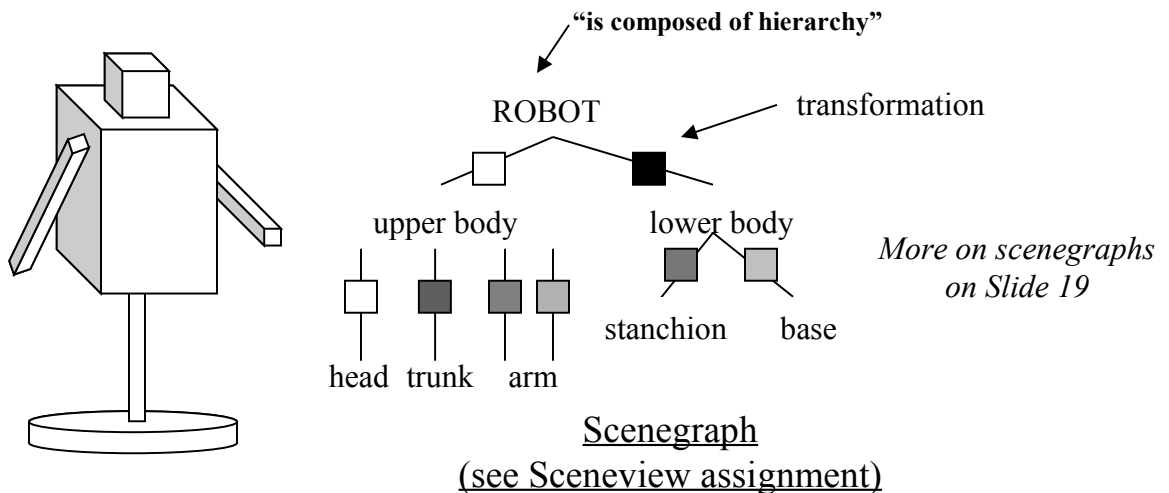


Geometric Transformations

How Are Geometric Transformations (T,R,S) Used in Computer Graphics?

- Object construction using assemblies/hierarchy of parts such as Sketchpad's masters and instances; leaves of scenegraph contain primitives



- Aid to realism
 - objects, camera use realistic motion
 - kinesthetic feedback as user manipulates objects or synthetic camera
- Synthetic camera/viewing
 - definition
 - normalization (from arbitrary view to canonical view)
- Note: Helpful applets
 - Experiment with these concepts on cs123 webpage: *Demos->Linear Algebra* and *Demos->Scenegraphs*

Useful concepts from Linear Algebra

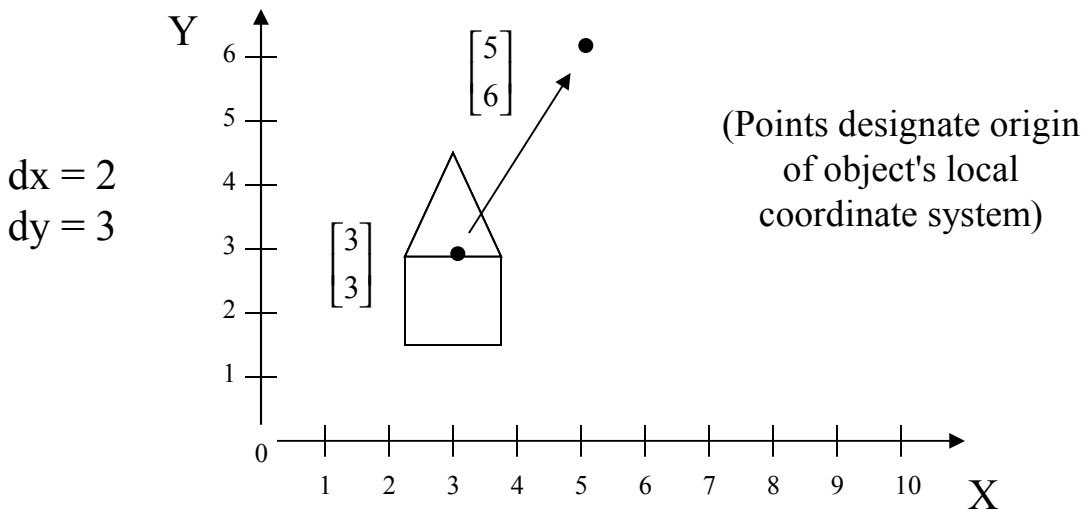
- 3D Coordinate geometry
- Vectors in 2 space and 3 space
- Dot product and cross product – definitions and uses
- Vector and matrix notation and algebra
- Multiplicative associativity
 - E.g. $A(BC) = (AB)C$
- Matrix transpose and inverse – definition, use, and calculation
- Homogeneous coordinates (x, y, z, \mathbf{w})

You will need to understand these concepts!

For a non-graphics example of the use of matrices and matrix/vector multiplication, see slide 24.

If you don't think you do, go to the
linear algebra help session (9/24 at 7-9pm)!

2D Translation



- Component-wise addition of vectors

$$\mathbf{v}' = \mathbf{v} + \mathbf{t} \quad \text{where} \quad \mathbf{v} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{v}' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} dx \\ dy \end{bmatrix}$$

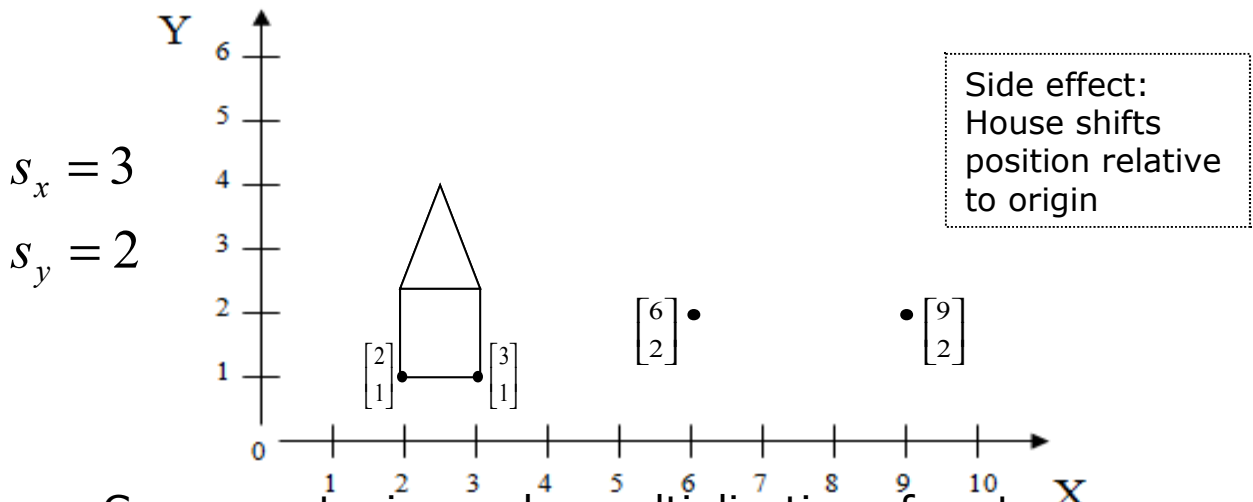
$$\text{and} \quad x' = x + dx$$

$$y' = y + dy$$

To move polygons: translate vertices (vectors) and redraw lines between them

- Preserves lengths (isometric)
- Preserves angles (conformal)
- Translation is thus "rigid-body"

2D Scaling



- Component-wise scalar multiplication of vectors

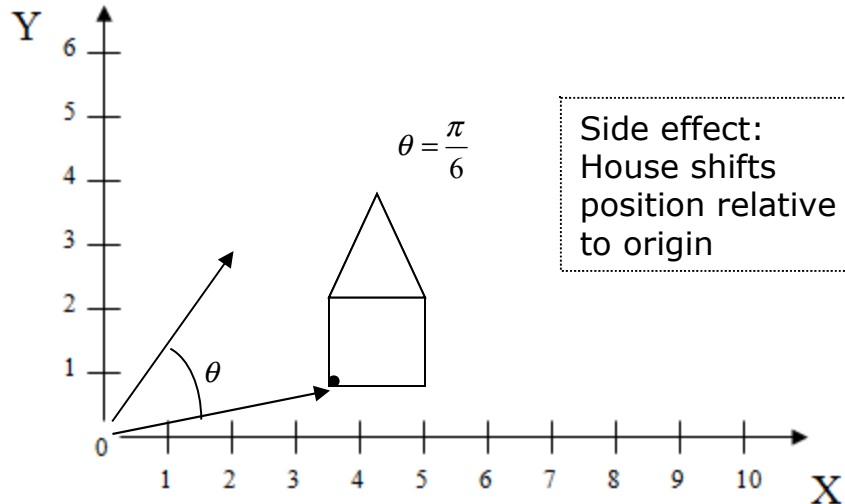
$$v' = Sv \quad \text{where} \quad v = \begin{bmatrix} x \\ y \end{bmatrix}, \quad v' = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

$$\text{and} \quad S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \quad \begin{matrix} x' = s_x x \\ y' = s_y y \end{matrix}$$

- Does not preserve lengths
- Does not preserve angles (except when scaling is uniform)

2D Rotation

Rotate by θ
about the
origin



$$v' = R_{\theta} v \quad \text{where} \quad v = \begin{bmatrix} x \\ y \end{bmatrix}, \quad v' = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

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

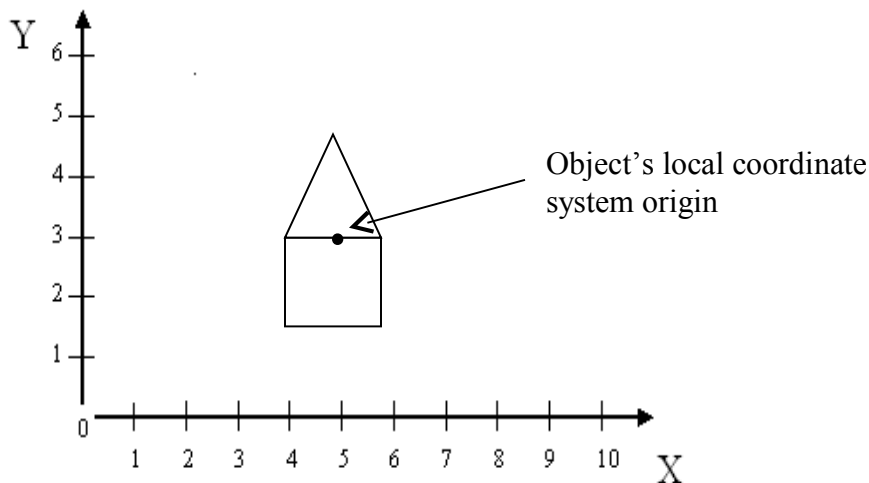
and $x' = x \cos \theta - y \sin \theta$
 $y' = x \sin \theta + y \cos \theta$

NB: A rotation by 0 angle, i.e. no rotation at all, gives us the identity matrix

- Proof by sine and cosine summation formulas
- Preserves lengths in objects, and angles between parts of objects
- Rotation is rigid-body

2D Rotation and Scale are Relative to Origin

- Suppose object is not centered at origin and we want to scale and rotate it.
- Solution: move to the origin, scale and/or rotate *in its local coordinate system*, then move it back.



- This sequence suggests the need to compose successive transformations...

Homogenous Coordinates

- Translation, scaling and rotation are expressed as:

$$\text{translation:} \quad v' = v + t$$

$$\text{scale:} \quad v' = Sv$$

$$\text{rotation:} \quad v' = Rv$$

- Composition is difficult to express
 - translation is not expressed as a matrix multiplication
- Homogeneous coordinates allows expression of all three transformations as 3x3 matrices for easy composition

$$P_{2d}(x, y) \rightarrow P_h(wx, wy, w), \quad w \neq 0$$

$$P_h(x', y', w), \quad w \neq 0$$

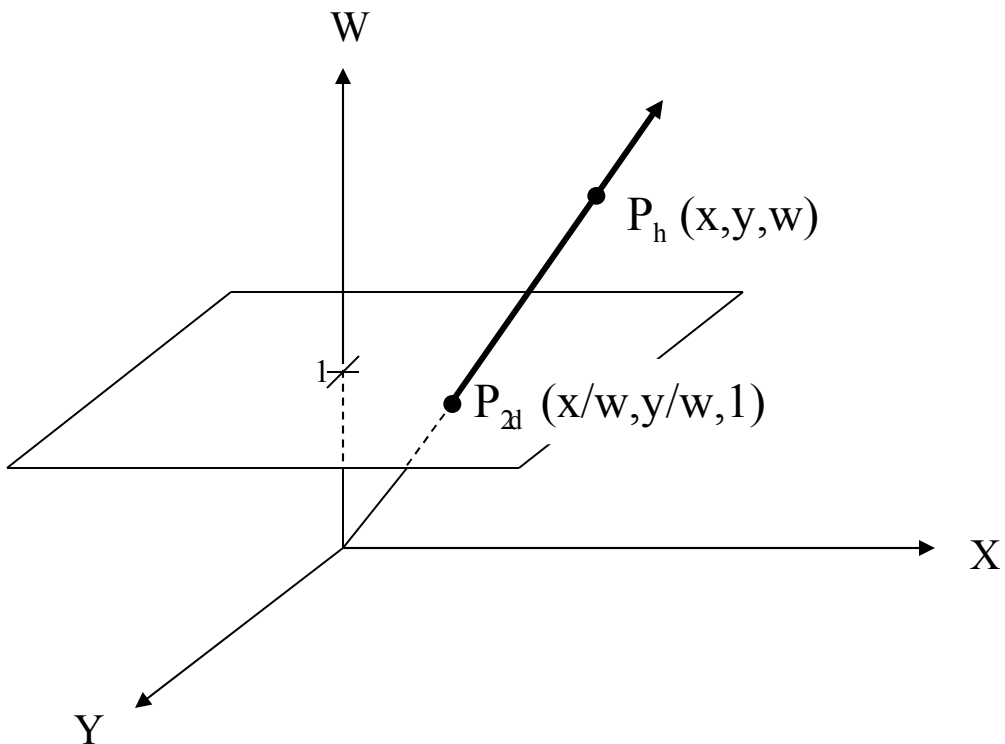
$$P_{2d}(x, y) = P_{2d}\left(\frac{x'}{w}, \frac{y'}{w}\right)$$

- w is 1 for affine transformations in graphics
- Note: $p = (x, y)$ becomes $p = (x, y, 1)$

This conversion does not transform p . It is only changing notation to show it can be viewed as a point on $w = 1$ hyperplane

What is $\begin{bmatrix} x \\ y \\ w \end{bmatrix}$?

- P_{2d} is intersection of line determined by P_h with the $w = 1$ plane



- Infinite number of points correspond to $(x, y, 1)$: they constitute the whole line (tx, ty, tw)

2D Homogeneous Coordinate Transformations (1/2)

- For points written in homogeneous coordinates,

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

translation, scaling and rotation relative to the origin are expressed homogeneously as:

$$T_{(d_x, d_y)} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \quad v' = T_{(d_x, d_y)} v$$

$$S_{(s_x, s_y)} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad v' = S_{(s_x, s_y)} v$$

$$R_{(\phi)} = \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad v' = R_{(\phi)} v$$

Examples

- Translate [1,3] by [7,9]

$$\begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 9 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 8 \\ 12 \\ 1 \end{bmatrix}$$

- Scale [2,3] by 5 in the X direction and 10 in the Y direction

$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 30 \\ 1 \end{bmatrix}$$

- Rotate [2,2] by 90° ($\pi/2$)

$$\begin{bmatrix} \cos(\pi/2) & -\sin(\pi/2) & 0 \\ \sin(\pi/2) & \cos(\pi/2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix}$$

2D Homogeneous Coordinate Transformations (2/2)

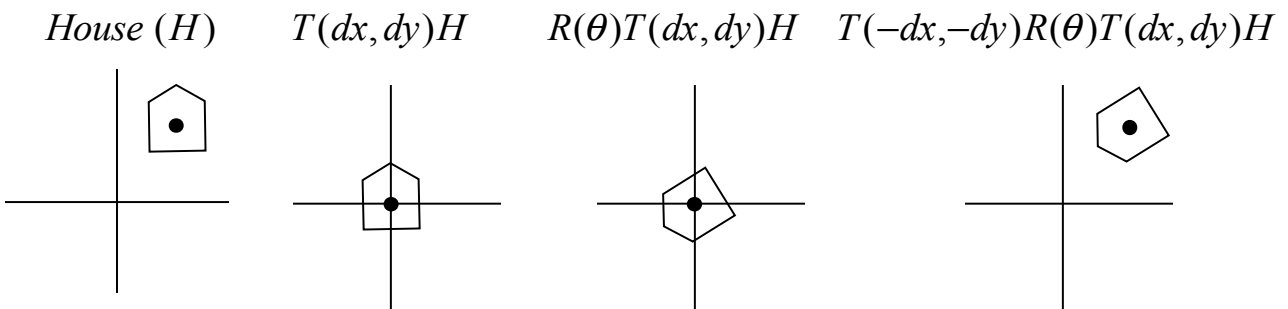
- Consider the rotation matrix:

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

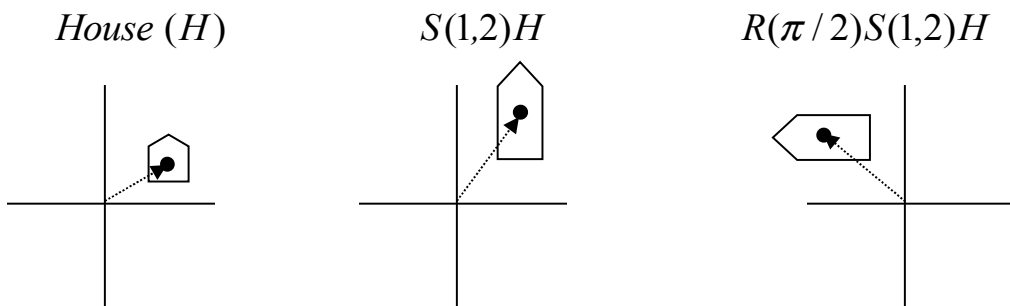
- The 2 x 2 submatrix columns are:
 - unit vectors (length=1)
 - perpendicular (dot product=0)
 - vectors into which X-axis and Y-axis rotate
- The 2 x 2 submatrix rows are:
 - unit vectors
 - perpendicular
 - vectors that rotate into X-axis and Y-axis
- Preserves lengths and angles of original geometry. Therefore, matrix is a “rigid body” transformation.

Matrix Compositions: Using Translation

- Avoiding unwanted translation when scaling or rotating an object not centered at origin:
 - translate object to origin, perform scale or rotate, translate back.



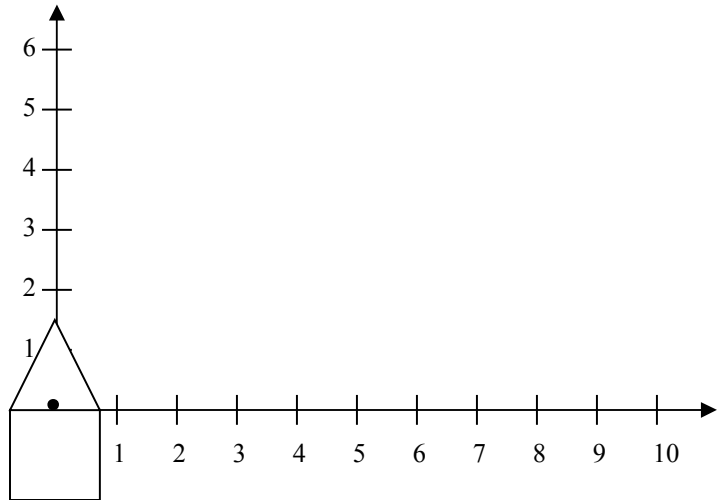
- How would you scale the house by 2 in “its” y and rotate it through 90° ?



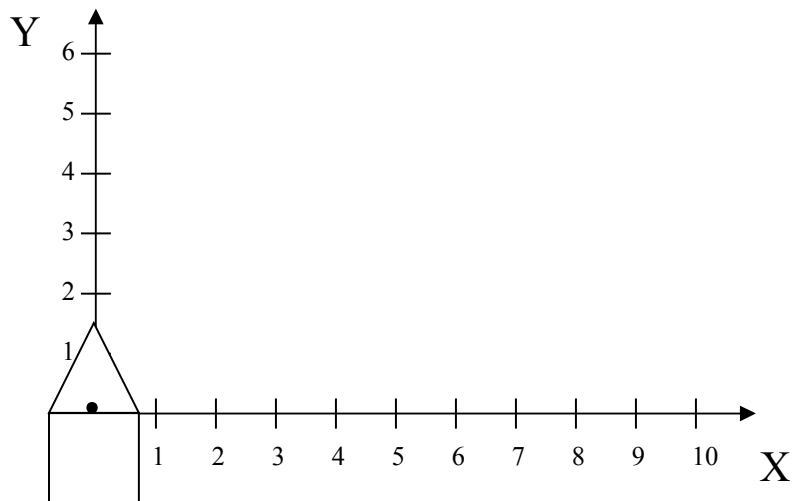
- Remember: matrix multiplication is not commutative! Hence order matters! (refer to the Transformation Game at *Demos->Scenegraphs*)

Matrix Multiplication is NOT Commutative

Translate by $x=6, y=0$ then rotate by 45°

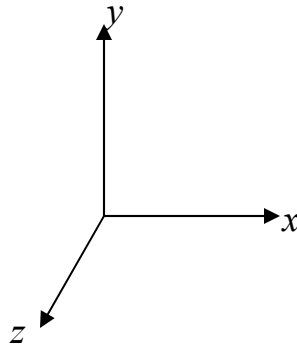


Rotate by 45° then translate by $x=6, y=0$



3D Basic Transformations (1/2)

(right-handed coordinate system)



- Translation
$$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Scaling
$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Basic Transformations (2/2)

(right-handed coordinate system)

- Rotation about X-axis

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

- Rotation about Y-axis

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

- Rotation about Z-axis

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

Homogeneous Coordinates

Some uses we'll be seeing later

- Placing sub-objects in parent's coordinate system to construct hierarchical scene graph
 - transforming primitives in own coordinate system
- View volume normalization
 - mapping arbitrary view volume into canonical view volume along z-axis
- Parallel (orthographic, oblique) and perspective projection
- Perspective transformation

Skew/Shear/Translate (1/2)

- “Skew” a scene to the side:

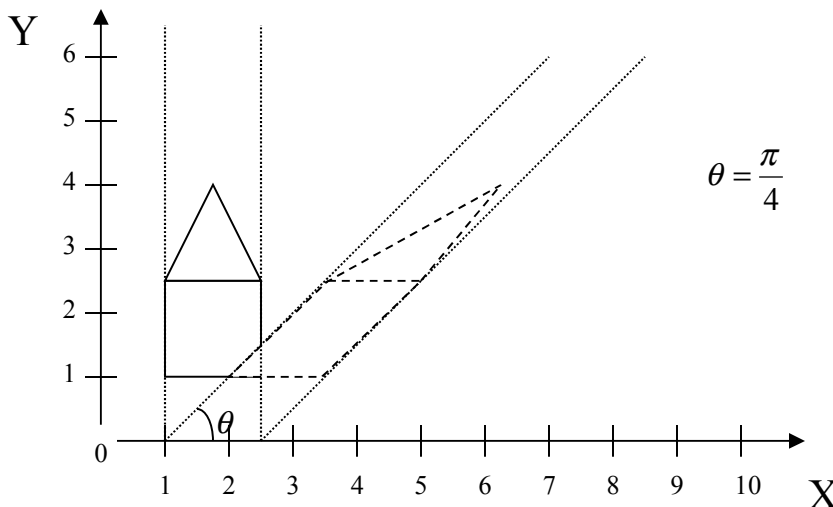
$$Skew_{\theta} = \begin{bmatrix} 1 & \frac{1}{\tan \theta} \\ 0 & 1 \end{bmatrix}$$

2D non-homogeneous

$$Skew_{\theta} = \begin{bmatrix} 1 & \frac{1}{\tan \theta} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2D homogeneous

- Squares become parallelograms - x coordinates skew to right, y coordinates stay same
- 90° between axes becomes θ
- Like pushing top of deck of cards to the side – each card shifts relative to the one below
- Notice that the base of the house (at $y=1$) remains horizontal, but shifts to the right.



NB: A skew of 0 angle, i.e. no skew at all, gives us the identity matrix, as it should

Transformations in Scene Graphs

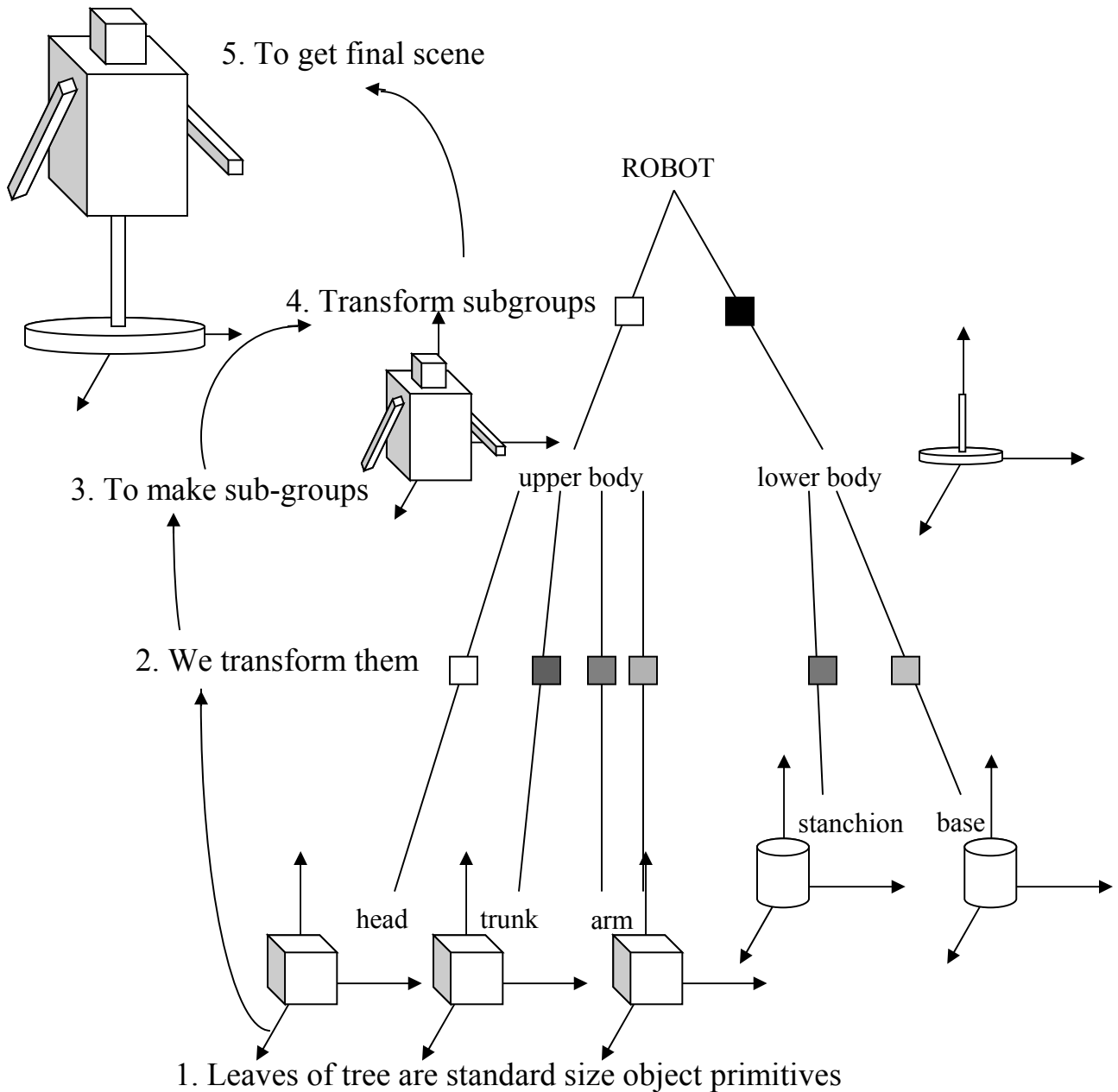
(1/3)

- 3D scenes are often stored in a directed acyclic graph (DAG) called a *scene graph*
 - WPF (implementation not accessible to the programmer)
 - Open Scene Graph (used in the Cave)
 - Sun's Java3D™
 - X3D™ (VRML™ was a precursor to X3D)
- Typical scene graph format:
 - **objects** (cubes, sphere, cone, polyhedra etc.)
 - stored as nodes (default: unit size at origin)
 - **attributes** (color, texture map, etc.) and **transformations** are also nodes in scene graph (labeled edges on slide 2 are an abstraction)

- For your assignments, use simplified format:
 - attributes stored as a component of each object node (no separate attribute node)
 - transform node affects its subtree, but not siblings
 - only leaf nodes are graphical objects
 - all internal nodes that are not transform nodes are group nodes

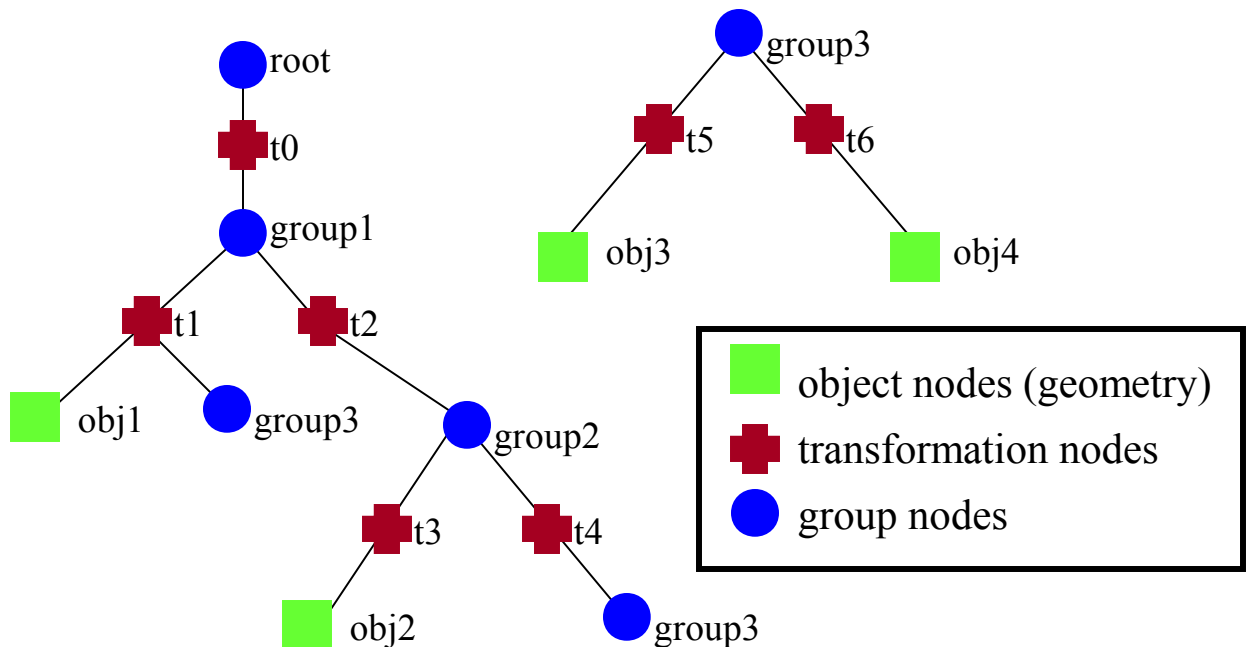
Transformations in Scene Graphs (2/3)

Closer look at Scenegraph from slide 2 ...



Transformations in Scene Graphs (3/3)

- Transformations affect all child nodes
- Sub-trees can be reused, called group nodes
 - instances of a group can have different transformations applied to them (e.g. group3 is used twice– once under t1 and once under t4)
 - must be defined before use



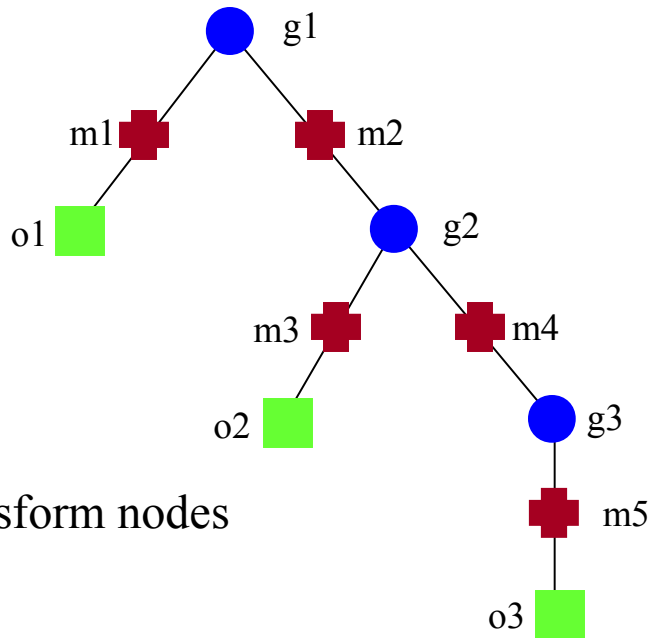
Composing Transformations in a Scene Graph (1/2)

- Transformation nodes contain at least a matrix that handles the transformation;
 - may also contain individual transformation parameters
 - refer to scene graph hierarchy applet by Dave Karelitz (URL on slide 2)

- To determine final composite transformation matrix (CTM) for object node:
 - compose all parent transformations during prefix graph traversal
 - exact detail of how this is done varies from package to package, so be careful

Composing Transformations in a Scene Graph (2/2)

- Example:



g: group nodes

m: matrices of transform nodes

o: object nodes

- for o1, CTM = m1
- for o2, CTM = m2* m3
- for o3, CTM = m2* m4* m5

- for a vertex v in o3, position in the world (root) coordinate system is:

$$\text{CTM } v = (m2 * m4 * m5)v$$

Short Linear Algebra

Digression: Vector and Matrix Notation, A non-Geometric Example (1/2)

Let's Go Shopping

- Need 6 apples, 5 cans of soup, 1 box of tissues, and 2 bags of chips
-
- Stores A, B, and C (East Side Market, Whole Foods, and Store 24) have following unit prices respectively

	1 apple	1 can of soup	1 box of tissues	1 bag of chips
East Side	\$0.20	\$0.93	\$0.64	\$1.20
Whole Foods	\$0.65	\$0.95	\$0.75	\$1.40
Store 24	\$0.95	\$1.10	\$0.90	\$3.50

A Non-Geometric Example (2/2)

- Shorthand representation of the situation (assuming we can remember order of items and corresponding prices):

- Column vector for quantities, q :
$$\begin{bmatrix} 6 \\ 5 \\ 1 \\ 2 \end{bmatrix}$$

- Row vector corresponding prices at stores (P):

store A (East Side)	[0.20 0.93 0.64 1.20]
store B (Whole Foods)	[0.65 0.95 0.75 1.40]
store C (Store 24)	[0.95 1.10 0.90 3.50]

What do I pay?

Let's calculate for each of the three stores.

- **Store A:**

$$\begin{aligned} \text{totalCost}_A &= \sum_{i=1}^4 P(A)_i q_i \\ &= (0.20 \cdot 6) + (0.93 \cdot 5) + (0.64 \cdot 1) + (1.20 \cdot 2) \\ &= (1.2 + 4.65 + 0.64 + 2.40) \\ &= 8.89 \end{aligned}$$

- **Store B:**

$$\begin{aligned} \text{totalCost}_B &= \sum_{i=1}^4 P(B)_i q_i = 3.9 + 4.75 + 0.75 + 2.8 \\ &= 12.2 \end{aligned}$$

- **Store C:**

$$\begin{aligned} \text{totalCost}_C &= \sum_{i=1}^4 P(C)_i q_i = 5.7 + 5.5 + 0.9 + 7 = 19.1 \\ & \quad i = 1 \end{aligned}$$

Using Matrix Notation

- Can express these sums more compactly:

$$P(All) = \begin{bmatrix} totalCost_A \\ totalCost_B \\ totalCost_C \end{bmatrix} = \begin{bmatrix} 0.20 & 0.93 & 0.64 & 1.20 \\ 0.65 & 0.95 & 0.75 & 1.40 \\ 0.95 & 1.10 & 0.90 & 3.50 \end{bmatrix} \begin{bmatrix} 6 \\ 5 \\ 1 \\ 2 \end{bmatrix}$$

- Determine totalCost vector by row-column multiplication
 - dot product is the sum of the pairwise multiplications
 - Apply this operation to rows of prices and column of quantities

$$\begin{bmatrix} a & b & c & d \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = ax + by + cz + dw$$

Book Chapter Readings

- Chapter 6 Section 6
 - Vectors/Matrices
- Chapter 9 Sections 1 through 9
 - Everything you wanted to know about transformations
- Chapter 10 Sections 1 through 4
 - Everything you need to know about transformations