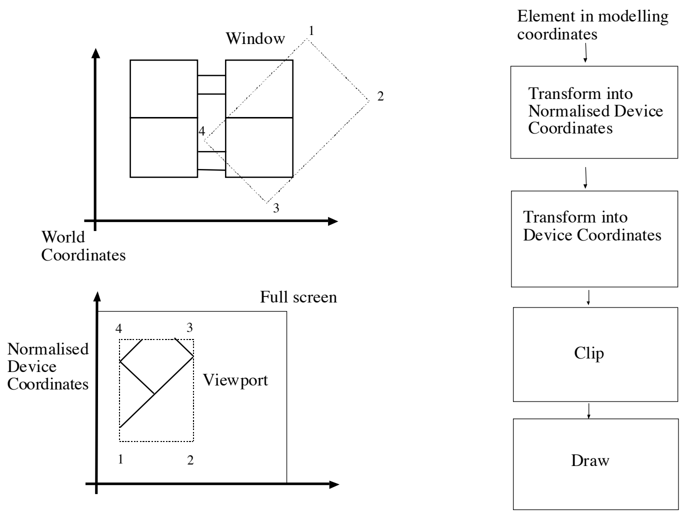
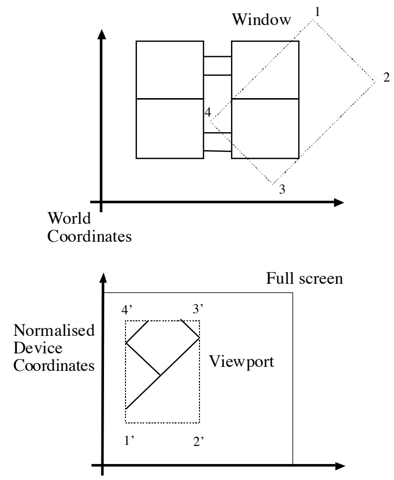



# Plan B

Work **directly** from the fact that the transform we seek must map 1 to 1', 2 to 2', 3 to 3'.

Details Optional

Plan B: Solve for the affine transformation directly.

- We know that this is an "affine" transform.
- In particular, the matrix we seek is:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$$

Details Optional

## More Details

- Consider the first mapping, $Mp_1 = q_1$
- $p_1 = (x_1, y_1, 1)^T$, $q_1 = (u_1, v_1, 1)^T$

$$\begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$$

$$\boxed{\begin{aligned} ax_1 + by_1 + c &= u_1 \\ dx_1 + ey_1 + f &= v_1 \end{aligned}}$$

## More Details

Details Optional

Write

$$ax_1 + by_1 + c = u_1$$

As

$$x_1 a + y_1 b + 1 \bullet c + 0 \bullet d + 0 \bullet e + 0 \bullet f = u_1$$

Notice that this gives one equation in the six unknowns

## More Details

Details Optional

Similarly, write

$$dx_1 + ey_1 + f = v_1$$

As

$$0 \bullet a + 0 \bullet b + 0 \bullet c + x_1 \bullet d + y_1 \bullet e + 1 \bullet f = u_1$$

Notice that this gives a second equation in the six unknowns

## More Details

Details Optional

- $Mp_1 = q_1$ gives first two rows
- $p_1 = (x_1, y_1, 1)^T$, $q_1 = (u_1, v_1, 1)^T$

$$\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix}$$

$$\boxed{\begin{matrix} ax_1 + by_1 + c = u_1 \\ dx_1 + ey_1 + f = v_1 \end{matrix}}$$

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} = \begin{pmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{pmatrix}$$

$Mp_2 = q_2$, $Mp_3 = q_3$ give other rows

## More Details

Details Optional

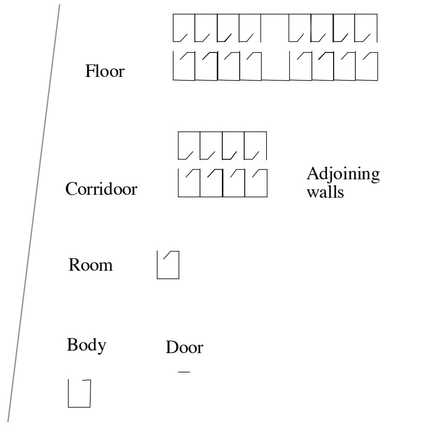
Final representation of six equations in six unknowns

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} = \begin{pmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{pmatrix}$$

This can be solved using standard methods

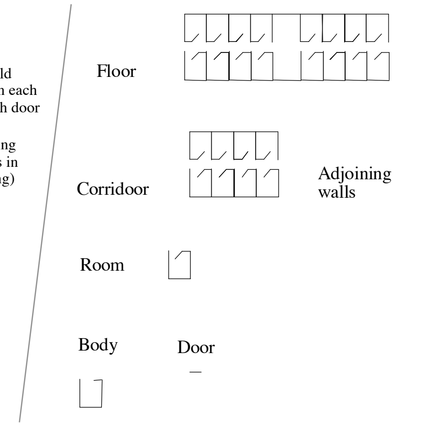
# Hierarchical modeling

- Consider constructing a complex 2d drawing: e.g. an animation showing the plan view of a building, where the doors swing open and shut.

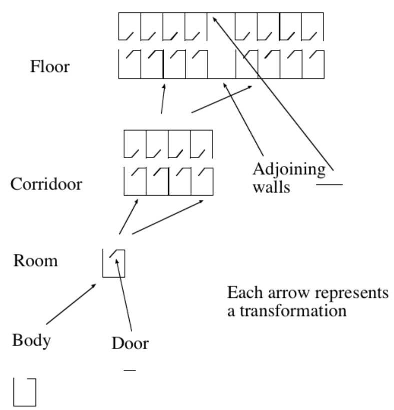



# Hierarchical modeling

- Options:
  - specify everything in world coordinate frame; but then each room is different, and each door moves differently.
  - Exploit similarities by using repeated copies of models in different places (instancing)




# Hierarchical modeling




# Hierarchical modeling

- Model form
  - Directed acyclic graph.
  - Each node consists of 0 or more objects (lines, polygons, etc).
  - Each edge is a transformation
- There can be many edges joining two nodes (e.g. in the case of the corridor - many copies of the same room model, each transformed differently).
- Every graphics API supports hierarchies - some directly (meaning you have to learn a language to express the model) some indirectly with a matrix stack

# Hierarchical modeling

Write the transformation from door coordinates to room coordinates as:

$$T_{room}^{door}$$

Then to render a door, use the transformation:

$$T_{device}^{world} T_{floor}^{corridoor} T_{corridoor}^{room} T_{room}^{door}$$

To render a body, use the transformation:

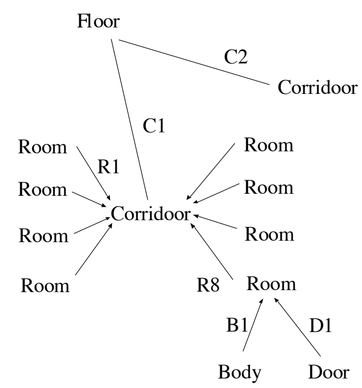
$$T_{device}^{world} T_{floor}^{corridoor} T_{corridoor}^{room} T_{room}^{body}$$

# Matrix stacks and rendering

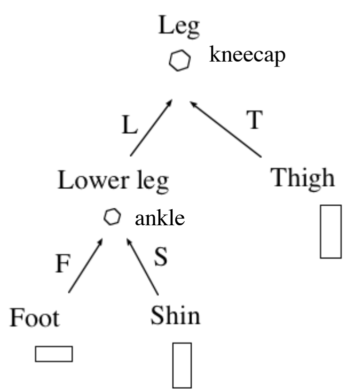
- Matrix stack:
  - Stack of matrices used for rendering
  - Applied in sequence.
  - Pop=remove last matrix
  - Push=append a new matrix
  - In previous example, body-device transformation comes from door-device transformation by popping door-room and pushing body-room

# Matrix stacks and rendering

- Root node has single edge with the world-to-device transformation.
- Algorithm for rendering a hierarchical model:
  - render (root)
- Recursive definition of render (node)
  - if node has object, render it
  - for each child:
    - push transformation
    - render (child)
    - pop transformation



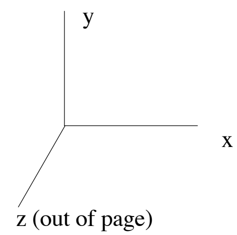

- Now to render door on first room in first corridor, stack looks like: W C1 R1 D1
- For efficiency we would store "running" products, IE, the stack contains: W, W*C1, W*C1*R1, W*C1*R1*D1.
- We do not need two copies of corridor, or 16 copies of body; we render one copy using 16 different transformations. This is known as instancing
- Animation requires care: if D1 is a single function of time, all doors will swing open and closed at the same time.



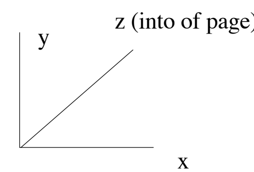

- Stack is W
- render kneecap
- Stack is W L
- render ankle
- Stack is W L F
- render foot
- Stack is W L S
- render shin
- Stack is W T
- render thigh

# Transformations in 3D

- Right hand coordinate system (conventional, i.e., in math)




- In graphics a LHS is sometimes also convenient (Easy to switch between them--later).




# Transformations in 3D

- Homogeneous coordinates now have four components - traditionally, (x, y, z, w)
  - ordinary to homogeneous: (x, y, z) -> (x, y, z, 1)
  - homogeneous to ordinary: (x, y, z, w) -> (x/w, y/w, z/w)
- Again, translation can be expressed as a multiplication.

# Transformations in 3D

- Translation:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

# 3D transformations

- Anisotropic scaling:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- Shear (one example):

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

# Rotations in 3D

- 3 degrees of freedom
- Orthogonal, with det(R)=1
- We can easily determine formulas for rotations about each of the axes
- For general rotations, there are many possible representations—we will use a **sequence** of rotations about coordinate axes.
- Sign of rotation follows the Right Hand Rule--point thumb along axis in direction of increasing ordinate--then fingers curl in the direction of positive rotation).

# Rotations in 3D

- About x-axis

$$M = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

# Rotations in 3D

- About y-axis

$$M = \begin{vmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

## Rotations in 3D

- About z-axis

$$M = \begin{vmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

## Commuting transformations

- If A and B are matrices, does AB=BA? Always? Ever?
- What if A and B are restricted to particular transformations?
- What about the 2D transformations that we have studied?
- How about if A and B are restricted to be on of the three specific 3D rotations just introduced, such as rotation about the Z axis?

**Answer**: In general AB != BA (matrix multiplication is not commutative). But if A and B are either translations or scalings, then multiplication is commutative. The same applies to rotations restricted to be about one of the 3 axis in 3D.

## Rotations in 3D

- About X axis

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

- 90 degrees about X axis

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

## Rotations in 3D

- About Y axis

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

- 90 degrees about Y axis

$$\begin{vmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

## Rotations in 3D

- 90 degrees about X then Y

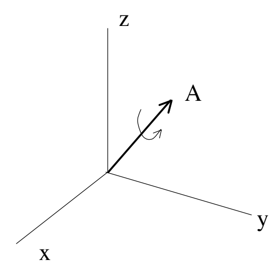
$$\underset{\text{Y rot}}{\begin{vmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}} \underset{\text{X rot}}{\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}} = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$\neq$$

- 90 degrees about Y then X

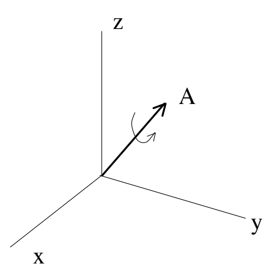
$$\underset{\text{X rot}}{\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}} \underset{\text{Y rot}}{\begin{vmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}} = \begin{vmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

## Rotation about an arbitrary axis




Strategy--rotate A to Z axis, rotate about Z axis, rotate Z back to A.

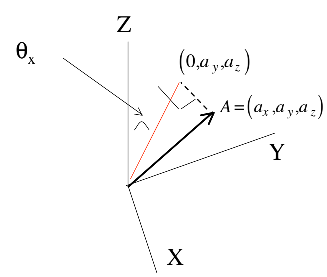
## Rotation about an arbitrary axis




Tricky part:
rotate A to Z axis

Two steps.
1) Rotate about x to xz plane
2) Rotate about y to Z axis.

## Rotation about an arbitrary axis

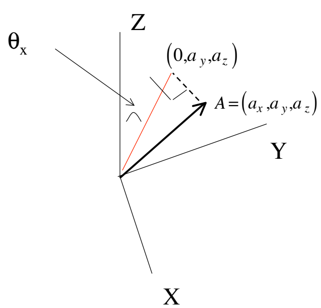



Tricky part:
rotate A to Z axis

Two steps.
1) Rotate about X to xz plane
2) Rotate about Y to Z axis.

As A rotates into the xz plane, its projection (shadow) onto the YZ plane (red line) rotates through the same angle which is easily calculated.

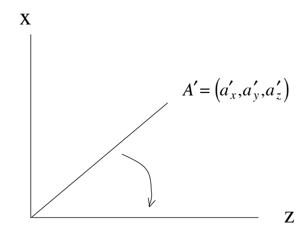
## Rotation about an arbitrary axis




$$d = \sqrt{a_y^2 + a_z^2}$$
$$\sin\theta_x = a_y / d$$
$$\cos\theta_x = a_z / d$$

No need to compute angles, just put sines and cosines into rotation matrices

## Rotation about an arbitrary axis




Apply $R_x(\theta_x)$ to A to get A'

$R_y(\theta_y)$ should be easy, but note that it is clockwise.

## Rotation about an arbitrary axis

Final form is

$$R_x(-\theta_x) R_y(-\theta_y) R_z(\theta_z) R_y(\theta_y) R_x(\theta_x)$$

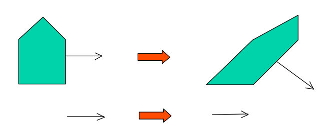
## Transforming the Normal Vector

- The normal to a polygon does **not** always transform in the same way as the points on the polygon
  - One case where is does: **Orthogonal**

    Somewhat intuitive

  - One case where it does not: **Shear**

$$M = \begin{vmatrix} 1 & a \\ 0 & 1 \end{vmatrix}$$



## Transforming the Normal Vector

- One way to find the transformed normal is to first transform the polygon, and then re-compute the normal.
- We can often save some time by computing the transform of the normal.