Homogenous Coordinates

- Represent 2D points by 3D vectors
- (x,y)-->(x,y,1)
- Now a multitude of 3D points (x,y,W) represent the same 2D point, (x/W, y/W, 1)
- Represent 2D transforms with 3 by 3 matrices
- Can now do translations
- Homogenous coordinates have other uses/advantages (later)

2D Translation in H.C.

$\mathbf{P}_{new} = \mathbf{P} + \mathbf{T}$

$(x', y') = (x, y) + (t_x, t_y)$

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

2D Scale in H.C.

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

2D Rotation in H.C.

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

Composition of Transformations

- If we use one matrix, M_1 for one transform and another matrix, M_2 for a second transform, then the matrix for the first transform followed by the second transform is simply $M_2 M_1$
- This generalizes to any number of transforms
- Computing the combined matrix **first** and then applying it to many objects, can save **lots** of computation

Composition Example

- Matrix for rotation about a point, P
- Problem--we only know how to rotate about the origin.

Composition Example

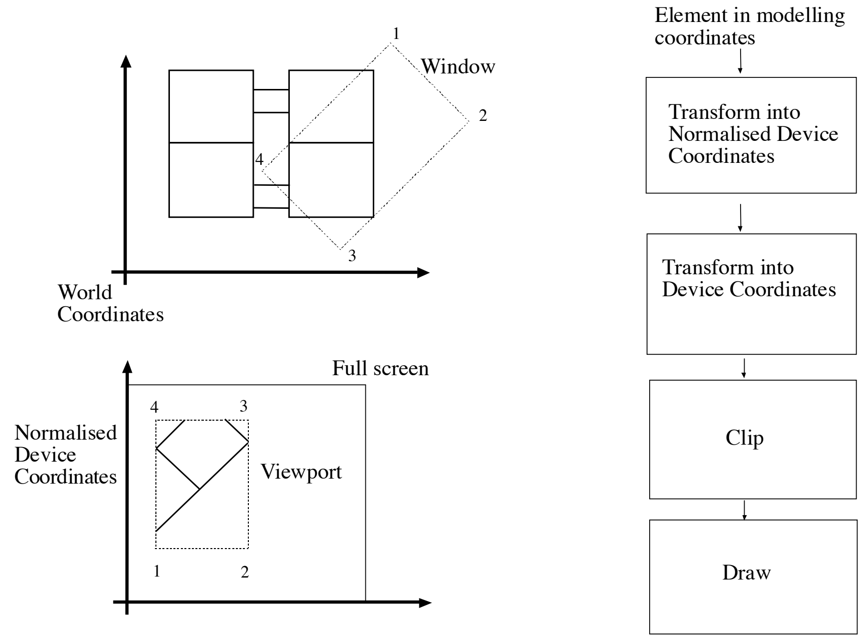
- Matrix for rotation about a point, P
- Problem--we only know how to rotate about the origin.
- Solution--translate to origin, rotate, and translate back

2D transformations (continued)

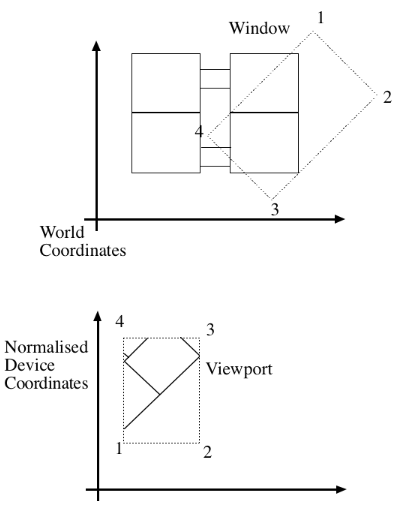
- The transformations discussed so far are invertable (why?). What are the inverses?

2D viewing

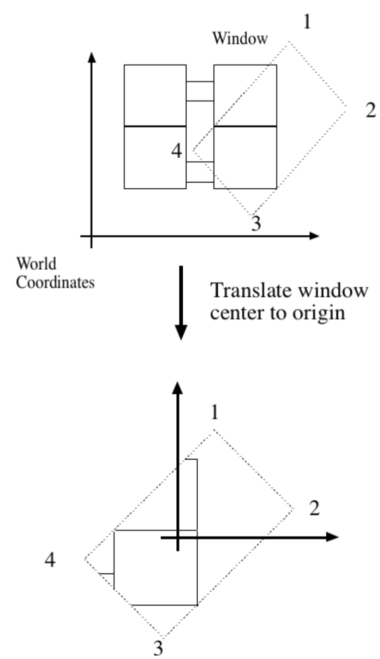
- Three coordinate systems are common in graphics
  - World coordinates or modeling coordinates - where the model is defined (meters, miles, etc.)
  - Normalized device coordinates; usually (0-1) in each variable.
  - Device coordinates: the actual coordinates of the pixels on the frame-buffer or the printer
- Need to construct transformations between coordinate systems
- Terminology:
  - window = region on drawing that will be displayed (rectangle)
  - viewport = region in NDC's/DC's where this rectangle is displayed (often simply entire screen).



Determining the transform



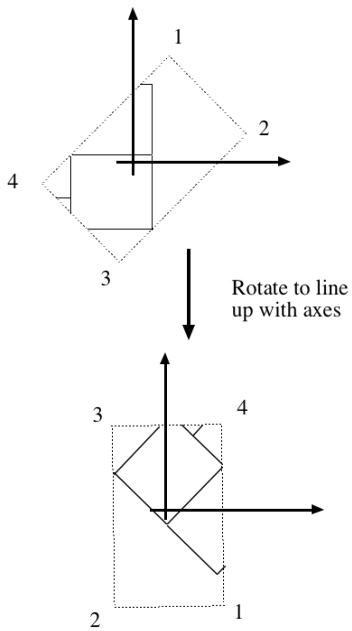
- **Plan A**: Consider this as a sequence of transformations in homogenous coords, then determine each element in closed form.
- **Plan B**: Compute numerically from point correspondences (not covered in detail in 2006)



- write (wx_i, wy_i) for coordinates of i'th point on window
- translation is:

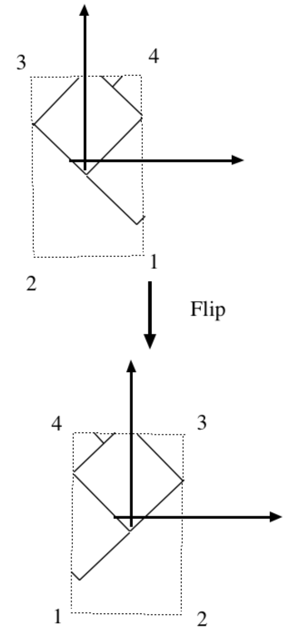
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\overline{wx} \\ 0 & 1 & -\overline{wy} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

(overbar denotes average over vertices, i.e., 1,2,3,4)



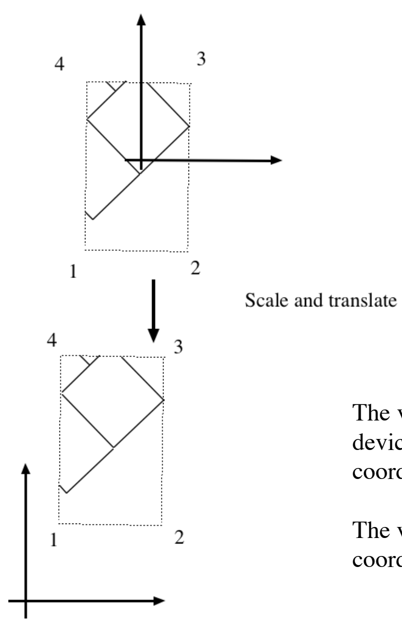
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

(Need to compute theta)



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

(Vertex order does not correspond, need to flip)



$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{w_{new}}{w_{old}} & 0 & \overline{x_{new}} \\ 0 & \frac{h_{new}}{h_{old}} & \overline{y_{new}} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

The variables labeled “new” are in either device coordinates or normalized device coordinates (depending on what you want).

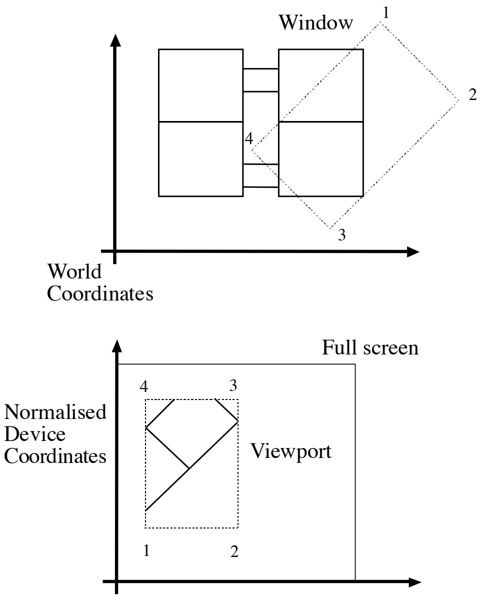
The variables labeled “old” are in world coordinates.

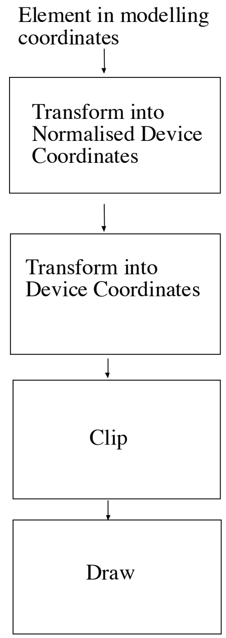
- Get overall transformation by multiplying transforms.
- This gives a single transformation matrix, whose elements are functions of window/viewport coordinates.

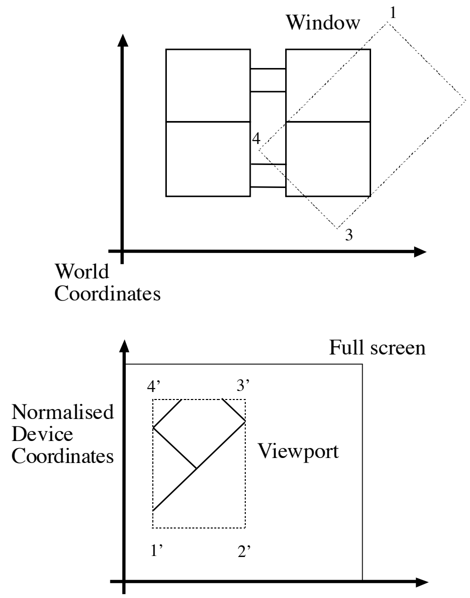
$x' = M_{(\text{translate origin to viewport cog, scale})}\, M_{(\text{flip})}\, M_{(\text{rotate})} M_{(\text{translate window cog->origin})} x$

NDC’s/DC’s | World coords

(cog==window center of gravity)







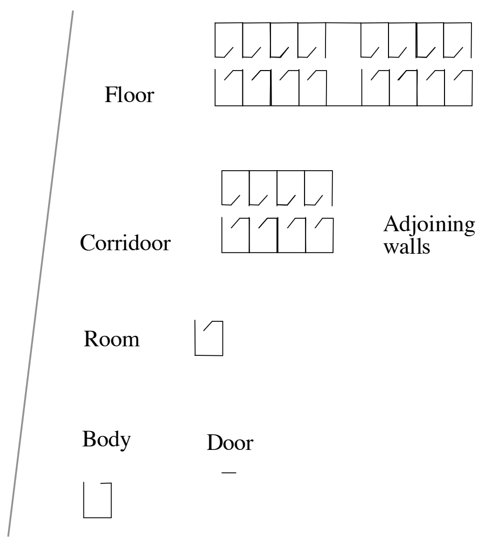
Plan B

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

Not covered in 2006

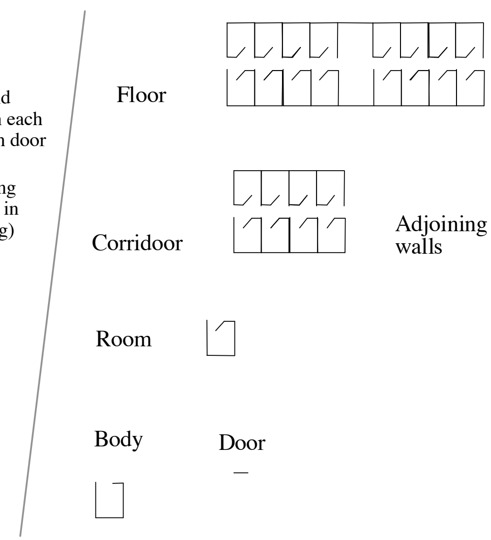
Hierarchical modeling

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

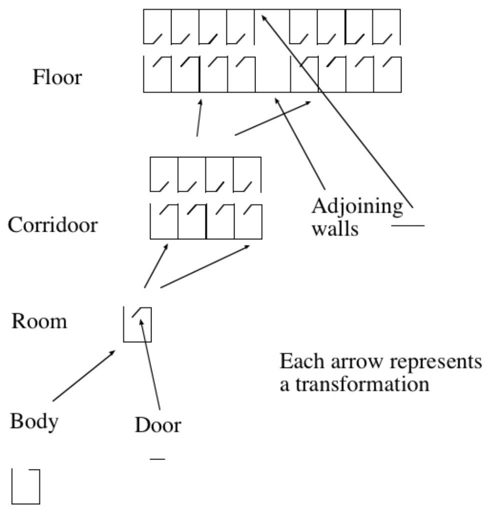


Hierarchical modeling

- 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:

$$T^{door}_{room}$$

Then to render a door, use the transformation:

$$T^{world}_{device} T^{corridoor}_{floor} T^{room}_{corridoor} T^{door}_{room}$$

To render a body, use the transformation:

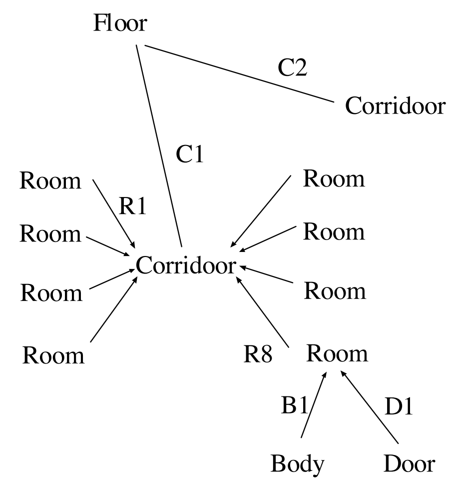
$$T^{world}_{device} T^{corridoor}_{floor} T^{room}_{corridoor} T^{body}_{room}$$

Matrix stacks and rendering

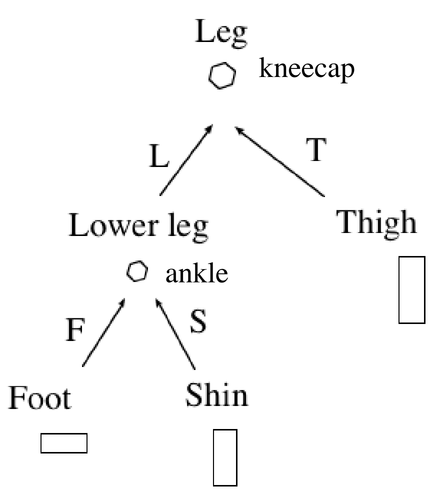
- Matrix stack:
  - Stack of matrices used for rendering
  - Applied in sequence.
  - Pop=remove last matrix
  - Push=append a new matrix
  - In previous example, body-device transformation comes from door-device transformation by popping door-room and pushing body-room

Matrix stacks and rendering

- Algorithm for rendering a hierarchical model:
  - stack is T_{device}^{root}
  - render (root)
- Recursive definition of render (node)
  - if node has object, render it
  - for each child:
    - push transformation
    - render (child)
    - pop transformation



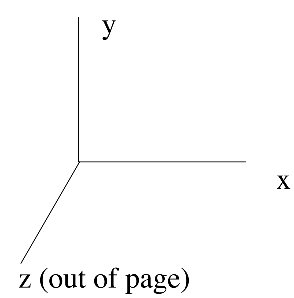
- Now to render door on first room in first corridor, stack looks like: W C1 R1 D1
- For efficiency we would store "running" products, IE, the stack contains: W, W*C1, W*C1*R1, W*C1*R1*D1.
- We do not need two copies of corridor, or 16 copies of body; we render one copy using 16 different transformations. 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.



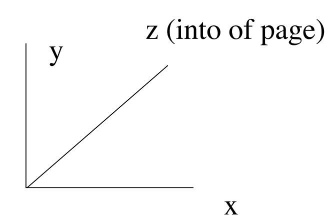
- Stack is W
- render kneecap
- Stack is W L
- render ankle
- Stack is W L F
- render foot
- Stack is W L S
- render shin
- Stack is W T
- render thigh

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, 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

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