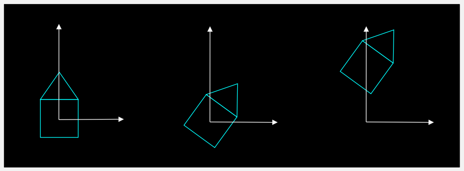
Combining Transformations

- order matters

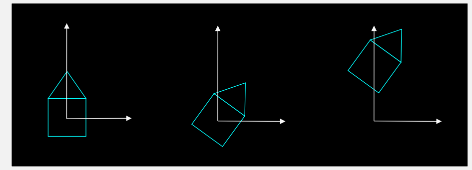


rotate, then translate

- in graphics systems, the current transformation applies to everything done after it
  - steps are written in reverse order
  - effect is as if the last transformation is applied first

translate(0,10)
rotate(-45)
draw house

Combining Transformations



rotate, then translate

- rotation

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

- followed by translation

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

Matrix Representation

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

$x' = s_x x$
$y' = s_y y$

$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

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

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

$x' = x + t_x$
$y' = y + t_y$

$?$

Matrix Representation

- to accommodate translation, switch to *homogeneous coordinates*
  - i.e. add a dimension (x,y) → (x,y,1)

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$$

Matrix Representation

x' = x + t_x
y' = y + t_y

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

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

Matrix Representation

shear

x' = x + sh_x y
y' = y

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

x' = x
y' = sh_y x + y

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

reflection

x' = x
y' = -y

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

x' = -x
y' = y

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

Combining Transformations

- matrix multiplication is associative
  - A(Bp) = (AB)p
- thus we can combine a bunch of transformations and then apply the result to points instead of having to apply each transformation separately
- this also explains why the order of transformations seems backwards

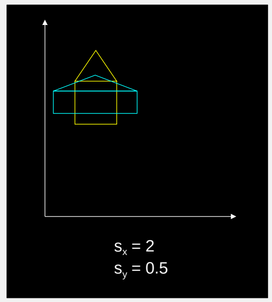
Inverses

- the *inverse* transformation undoes the transformation
  - translate by (t_x,t_y)
    - → translate by (-t_x,-t_y)
  - scale by (s_x,s_y)
    - → scale by (1/s_x,1/s_y)
  - rotate by θ
    - → rotate by -θ

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

General Scaling and Rotation

- “general” refers to a fixed point other than (0,0)



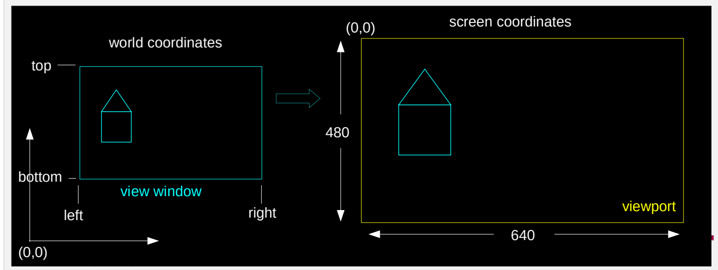
- strategy
  - translate desired fixed point to origin
    - T(-px,-py)
  - do scale/rotation
  - translate back
    - T(px,py)

→ T(px,py) S(2,0.5) T(-px,-py)

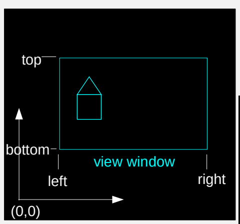
Application of Transforms – Viewing

- the display window on the screen (*viewport*) is in *screen coordinates* (SC)
- objects in the scene are defined in *world coordinates* (WC)

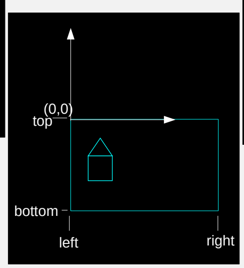
→ need to transform WC to SC
  - first, define the *view window* (in WC)



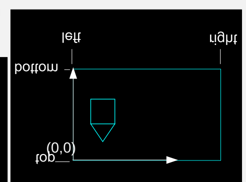
Window-to-Viewport...



translate top left corner of view window to origin



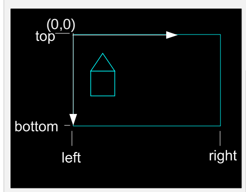
reflect around x axis



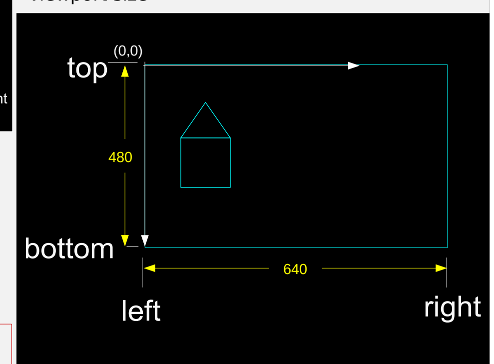
the house looks upside down, but that is because we are still showing (0,0) at the bottom – the coordinates are correct (the former top edge is at 0 and the former bottom edge has a positive y coordinate value) so it is just a matter of drawing with (0,0) at the top to get the correct picture

this is the order of application, not of writing the commands!

...Window-to-Viewport



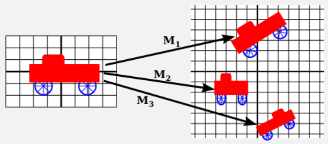
scale view window to viewport size



this is the order of application, not of writing the commands!

Application of Transforms – Modeling

- defining objects in WC is more convenient than SC, but why stop there?
  - define a canonical version of an object in *object coordinates* (OC) and then apply a *modeling transformation* to place it into WC



- advantages
  - simplifies modeling
  - saves effort if objects are repeated in the scene

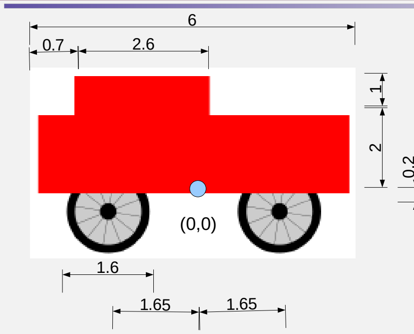
Viewing Pipeline

- viewing pipeline
  - OC → modeling transformation → WC → window-to-viewport → SC

Hierarchical Modeling

- defining objects in OC is more convenient than WC, but why stop there?
  - define a canonical version of each primitive and then apply transformation(s) to place it into OC for the object
- advantages
  - allows graphics libraries to provide primitives without zillions of parameters

Hierarchical Modeling

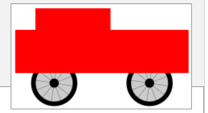


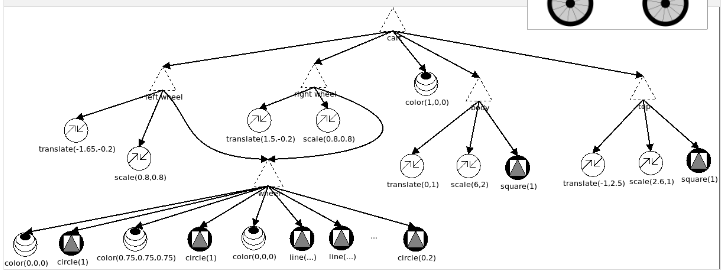
available primitives:
– draw filled square with specified side length, centered at (0,0)
– draw filled circle with specified radius, centered at (0,0)
– draw line between two endpoints

start with drawing a wheel in a convenient coordinate system (centered at (0,0), radius 1)
then size and place body and wheels to build the cart

Scene Graphs

- the hierarchical structure of a scene is captured in a *scene graph*





  - can be *implicit* through method calls and the program call stack
  - can be *explicit* with an actual data structure