Linear Transformations

- A linear function f(x) satisfies (by definition):

$$f(ax+by)=af(x)+bf(y)$$

- Note that "x" can be an abstract entity (e.g. a vector)–as long as addition and multiplication by a scalar are defined.

- Now consider the linear transformation of a point on a line segment connecting two points, **x** and **y**.
- Recall that in parametric form, that point is: $t\mathbf{x}+(1-t)\mathbf{y}$
- The transformed point is: $f(t\mathbf{x}+(1-t)\mathbf{y})=tf(\mathbf{x})+(1-t)f(\mathbf{y})$
- Notice that is a point on the line segment from the point $f(\mathbf{x})$ to the point $f(\mathbf{y})$
- This shows that a linear transformation maps line segments to line segments

---

[H&B chapter 5]

2D Transformations

- Represent **linear** transformations by matrices
- To transform a point, represented by a vector, multiply the vector by the appropriate matrix.
- Recall the definition of matrix times vector:

$$\begin{pmatrix} a_{11}x+a_{12}y \\ a_{21}x+a_{22}y \end{pmatrix}=\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$$

- Algebra reveals that matrix multiplication linear
- In particular., if we define f(**x**)=M • **x**, where M is a matrix and **x** is a vector, then

$$\begin{aligned} f(a\mathbf{x}+b\mathbf{y}) &= M(a\mathbf{x}+b\mathbf{y}) \\ &= aM\mathbf{x}+bM\mathbf{y} \\ &= af(\mathbf{x})+bf(\mathbf{y}) \end{aligned}$$

- Where the middle step can be verified using algebra (next slide)

Proof that matrix multiplication is linear

$$M(a\mathbf{x}+b\mathbf{y}) = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} ax_1 + by_1 \\ ax_2 + by_2 \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}ax_1 + a_{11}by_1 + a_{12}ax_2 + a_{12}by_2 \\ a_{21}ax_1 + a_{21}by_1 + a_{22}ax_2 + a_{22}by_2 \end{pmatrix}$$

$$= a\begin{pmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{pmatrix} + b\begin{pmatrix} a_{11}y_1 + a_{12}y_2 \\ a_{21}y_1 + a_{22}y_2 \end{pmatrix}$$

$$= aM\mathbf{x} + bM\mathbf{y}$$

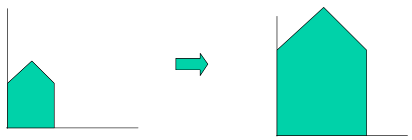
[H&B chapter 5]

2D Transformations of objects

- To transform line segments, transform endpoints
- To transform polygons, transform vertices

2D Transformations

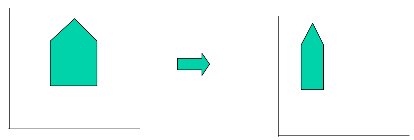
- Scale (stretch) by a factor of k



$M = \begin{vmatrix} k & 0 \\ 0 & k \end{vmatrix}$ (k = 2 in the example)

2D Transformations

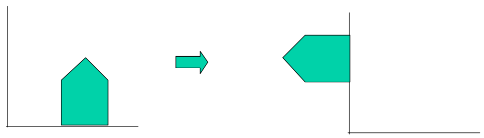
- Scale by a factor of (S_x, S_y)



$M = \begin{vmatrix} S_x & 0 \\ 0 & S_y \end{vmatrix}$ (Above, S_x= 1/2, S_y =1)

2D Transformations

- Rotate around origin by θ (Orthogonal)



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

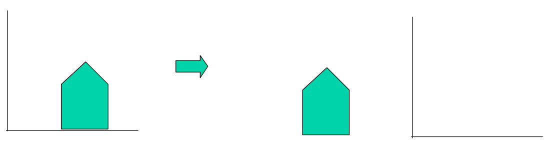
(Above, $\theta=90^o$)

Orthogonal Matrices

- Defined by: $M^TM = MM^T = I$ (M is square)
- Its inverse is its transpose
- Columns of M are orthonormal
  - of unit length
  - orthogonal to each other

2D Transformations

- Reflection through y axis
  - Not the same as rotating about y axis in 3D
  - Orthogonal

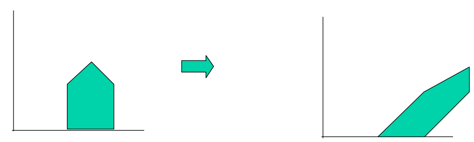


$$M = \begin{vmatrix} -1 & 0 \\ 0 & 1 \end{vmatrix}$$

Reflect over x axis is ?

2D Transformations

- Shear along x axis

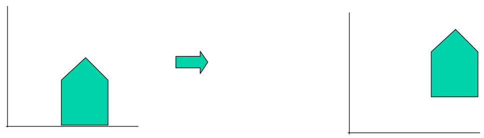


$$M = \begin{vmatrix} 1 & a \\ 0 & 1 \end{vmatrix}$$

Shear along y axis is ?

2D Transformations

- Translation $(\mathbf{P}_{new} = \mathbf{P} + \mathbf{T})$



M =?

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

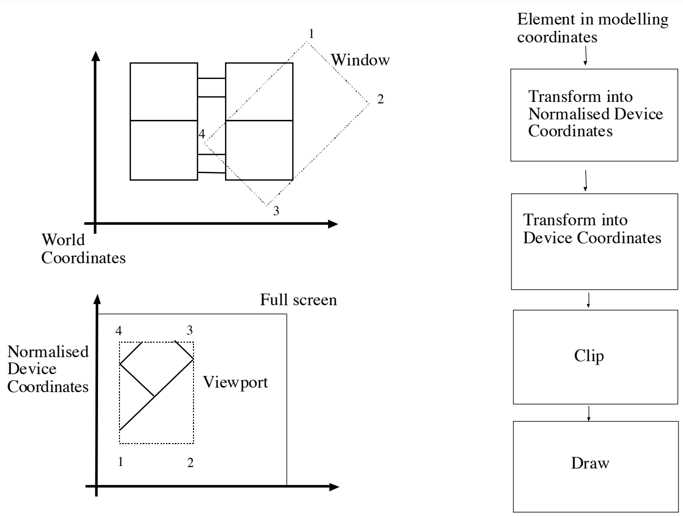
- Matrix for rotation about a point, P
- Problem--we only know how to rotate about the origin.
- Solution--translate to origin, rotate, and translate back

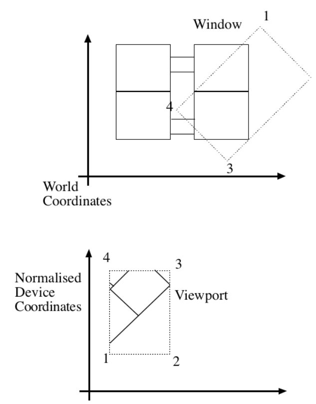
2D transformations (continued)

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

2D viewing

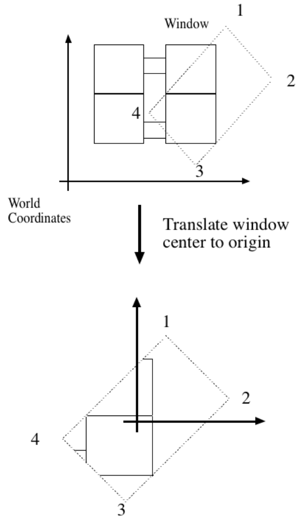
- Three coordinate systems are common in graphics
  - World coordinates or modeling coordinates - where the model is defined (meters, miles, etc.)
  - Normalized device coordinates; usually (0-1) in each variable.
  - Device coordinates: the actual coordinates of the pixels on the frame-buffer or the 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

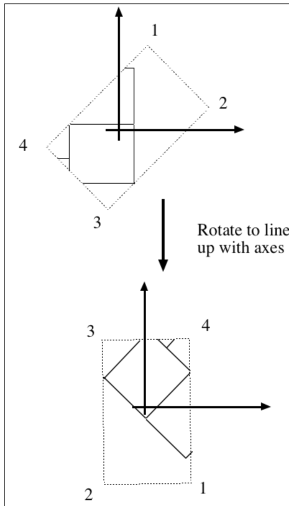
- **Plan A**: Consider this as a sequence of transformations in homogenous coords, then determine each element in closed form.
- **Plan B**: Compute numerically from point correspondences.



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

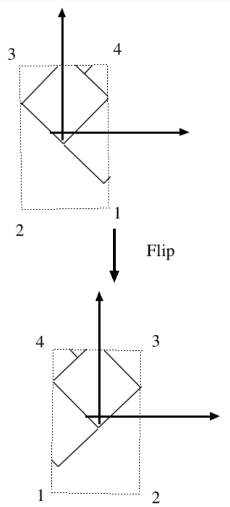
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\overline{wx} \\ 0 & 1 & -\overline{wy} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

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



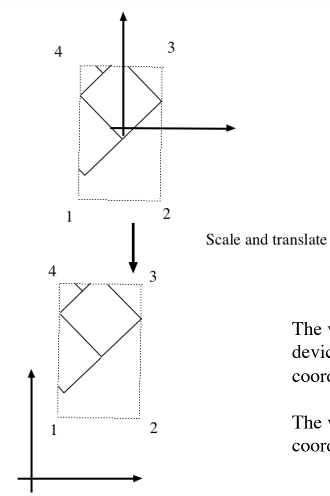
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

(Need to compute matrix elements)



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

(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)