2D Transformation

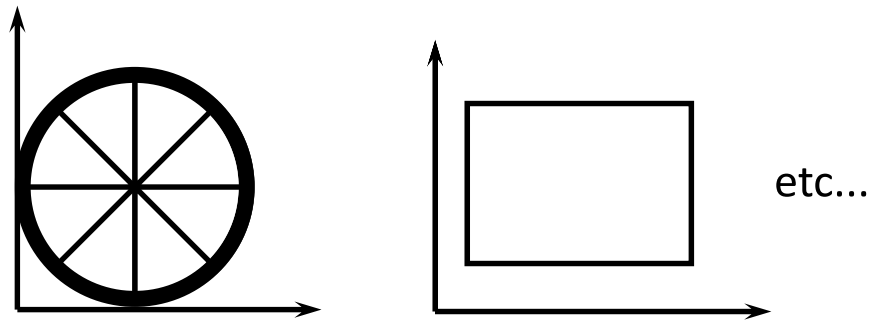
Dr. Mousumi Dutt

Transforming Pictures

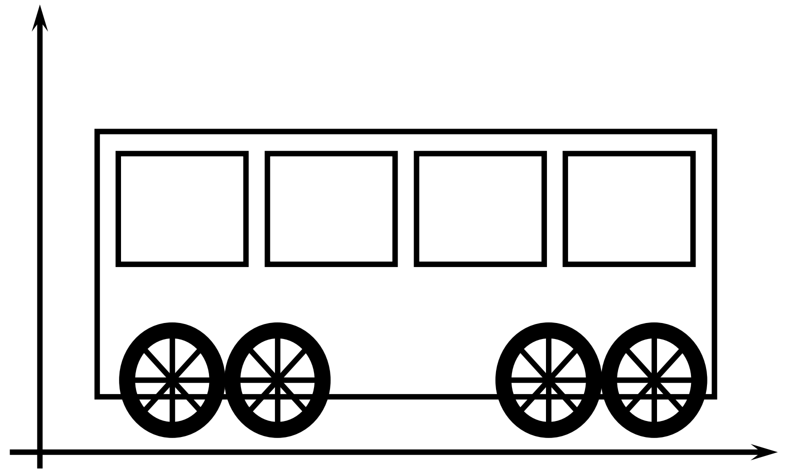
- Sometimes objects exhibit certain symmetries, so only a part of it needs to be described, and the rest constructed by reflecting, rotating and translating the original part
- A designer may want to view and object from different vantage points, by rotating the object, or by moving a “synthetic camera” viewpoint.
- In animation, one or more objects must move relative to one another, so that their local co-ordinate systems must be shifted and rotated as the animation proceeds.

Example 1

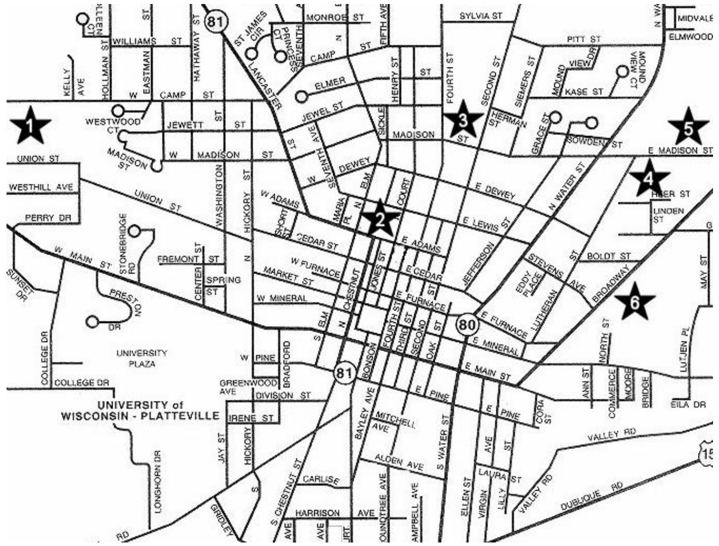
- Object parts defined in a local co-ordinate system:



- Larger objects are then "assembled" by duplicating and transforming each of the constituent parts:

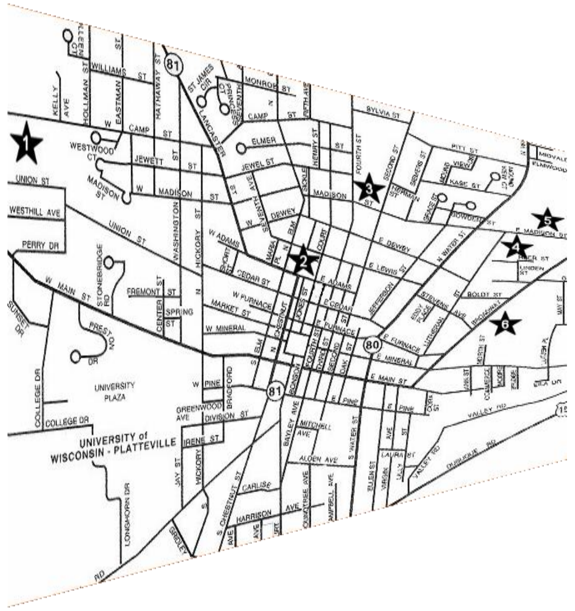


Example 2

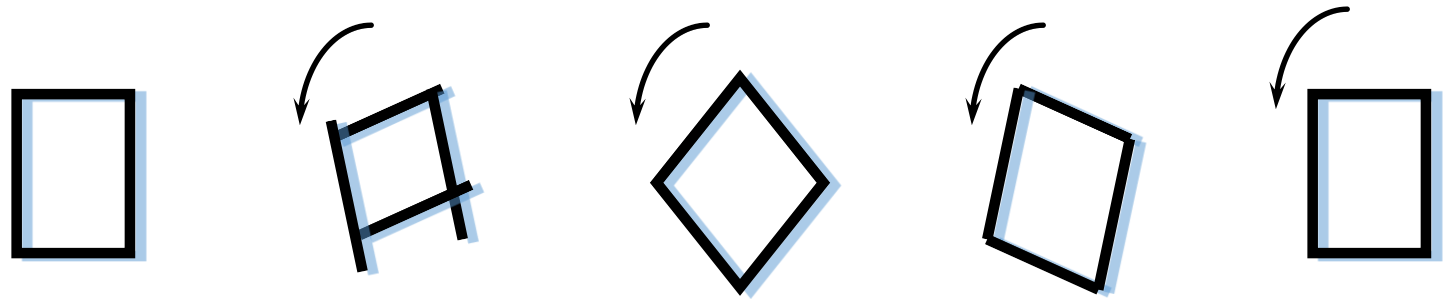


UNIVERSITY of WISCONSIN - PLATTEVILLE
UNIVERSITY PLAZA
UNION ST
W MAIN ST
WASHINGTON
HICKORY ST
E MADISON ST
VALLEY RD
DUBUQUE RD
1
2
3
4
5
6
81
80

CAMP ST
JEWETT ST
W MADISON ST
UNION ST
WASHINGTON
HICKORY ST
W ADAMS
SHORT
CEDAR
FREMONT ST



Example 3


5 steps of a “rotating cube” animation

- At each frame of the animation, the object is transformed, in this case by a rotation.
- It could also be transformed by changing its size (scaling), or its shape (deformation), or its location (translation).
- Further animation effects can be achieved by not changing the object, but the way it is viewed zooming and panning the viewing window

Transformations

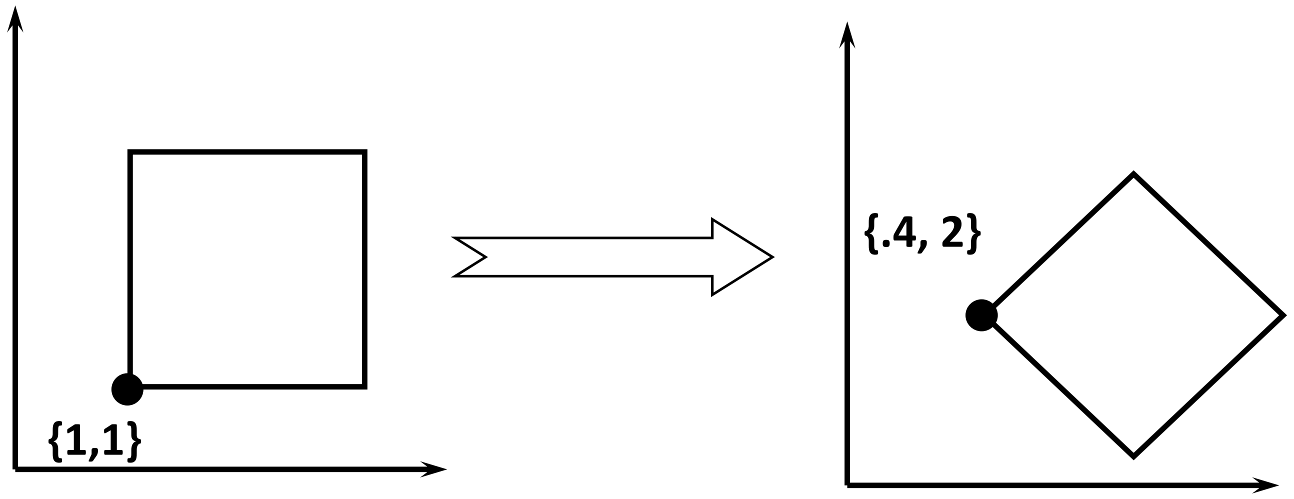
- A transformation on an object is an operation that changes how the object is finally drawn to screen

- There are two ways of understanding a transformation

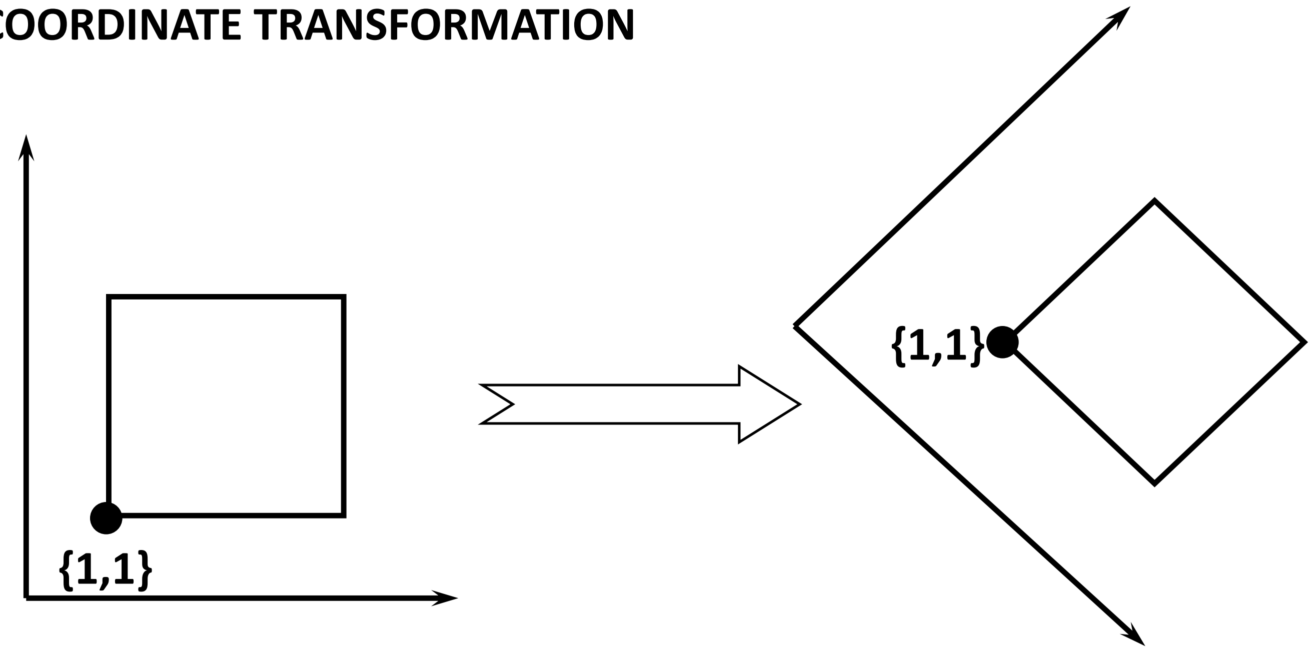
  - An **Object Transformation** alters the coordinates of each point according to some rule, leaving the underlying coordinate system unchanged

  - A **Coordinate Transformation** produces a different coordinate system, and then represents all original points in this new system

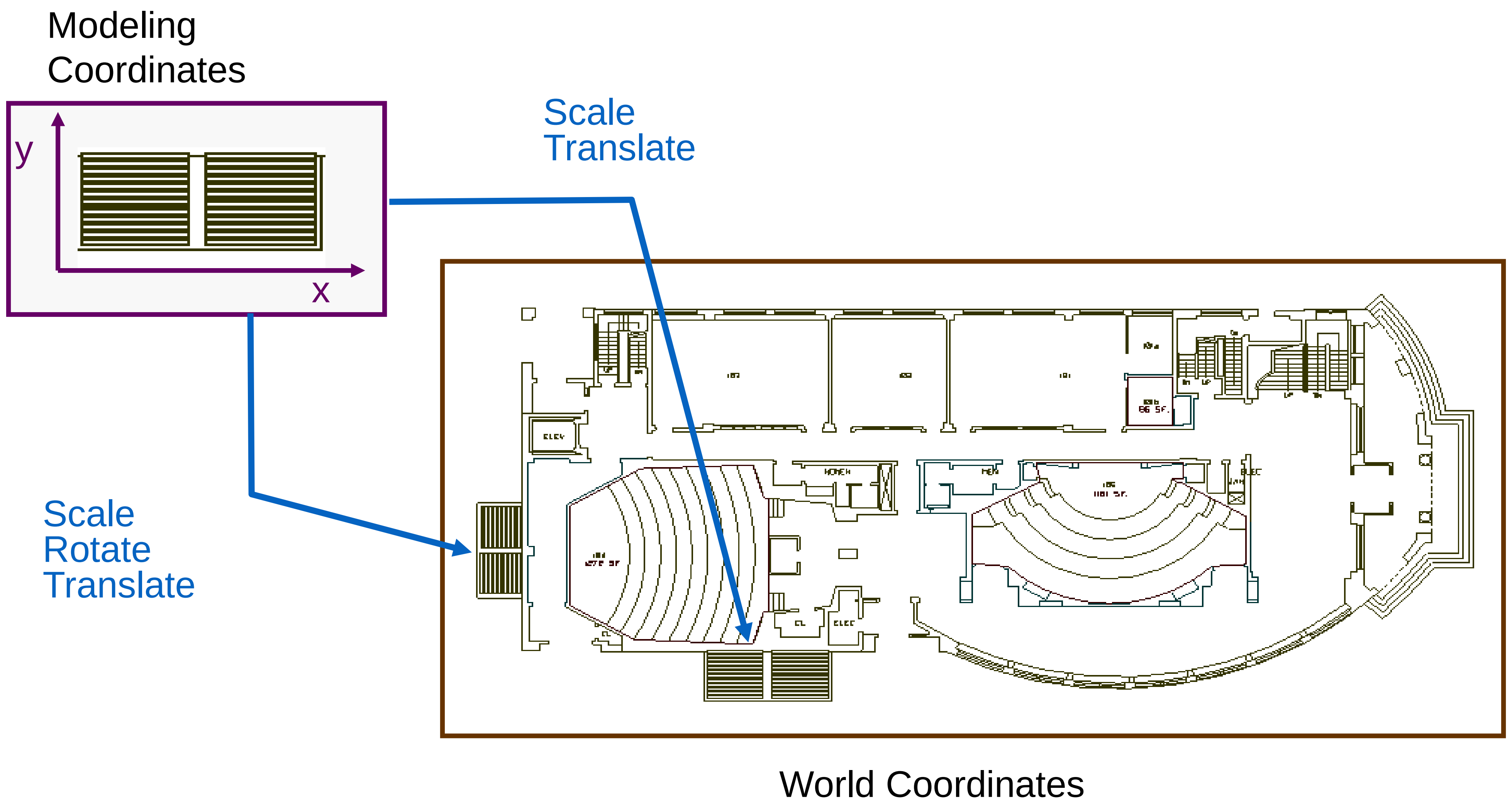
Object Transformations



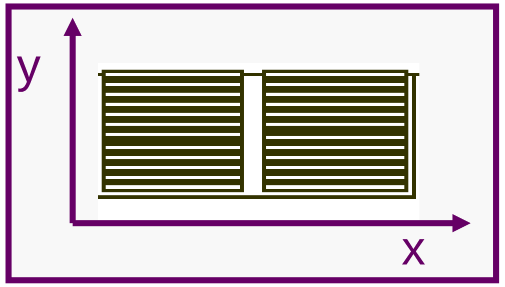
Example: COORDINATE TRANSFORMATION

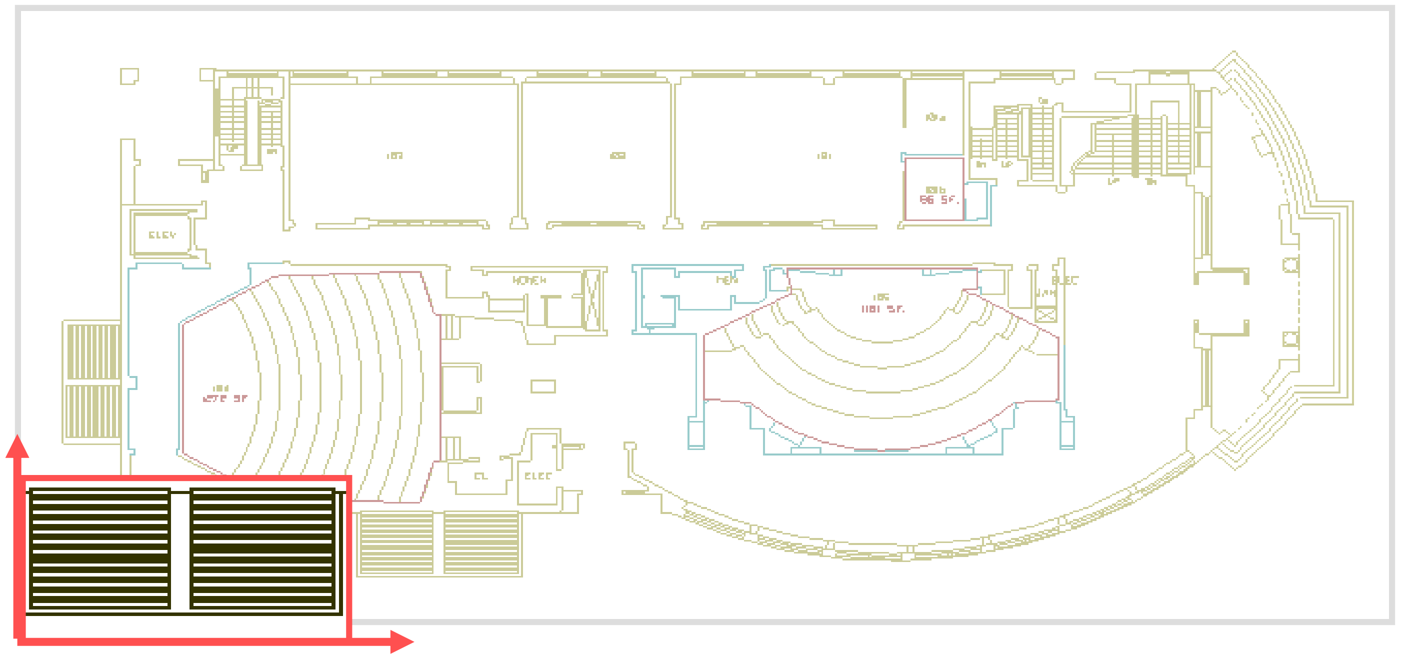


Object Transformations



Object Transformations

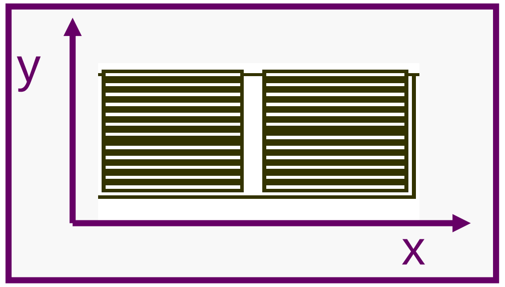




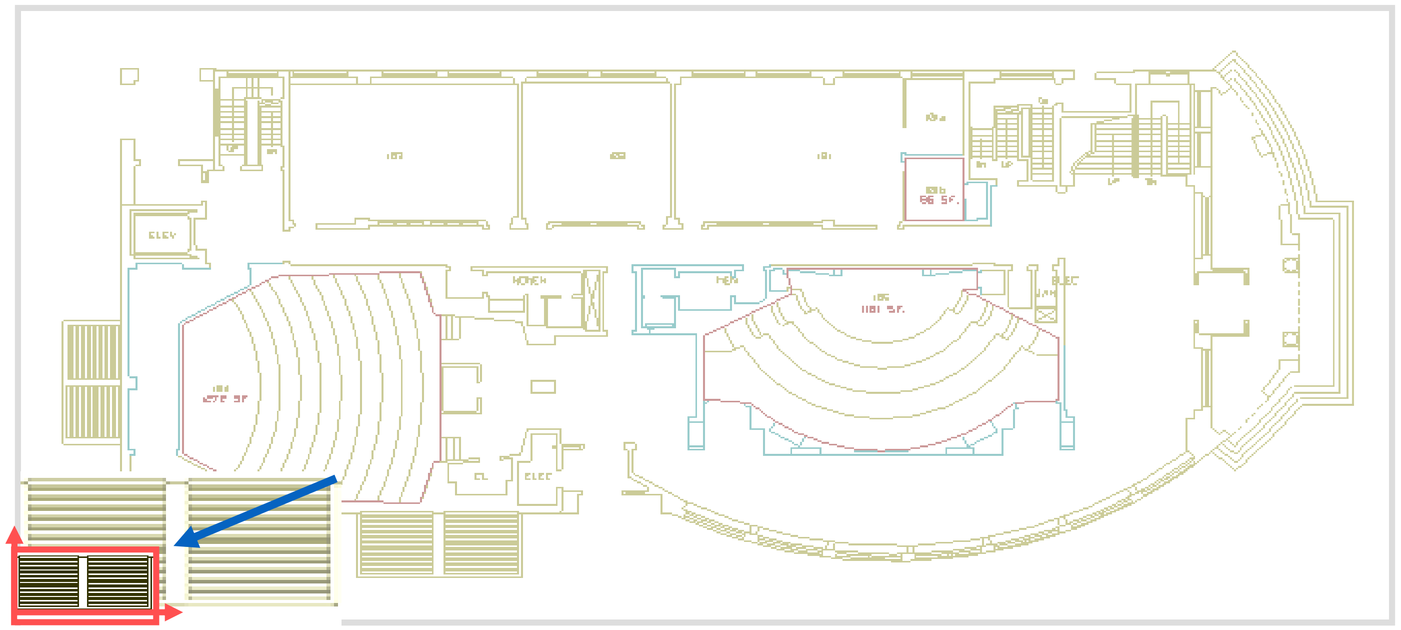
Initial location at (0, 0) with x- and y-axes aligned

Object Transformations

Modeling Coordinates

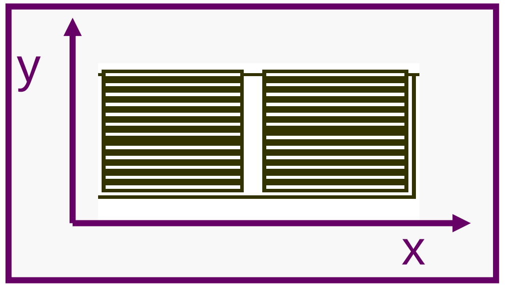


Scale .3, .3

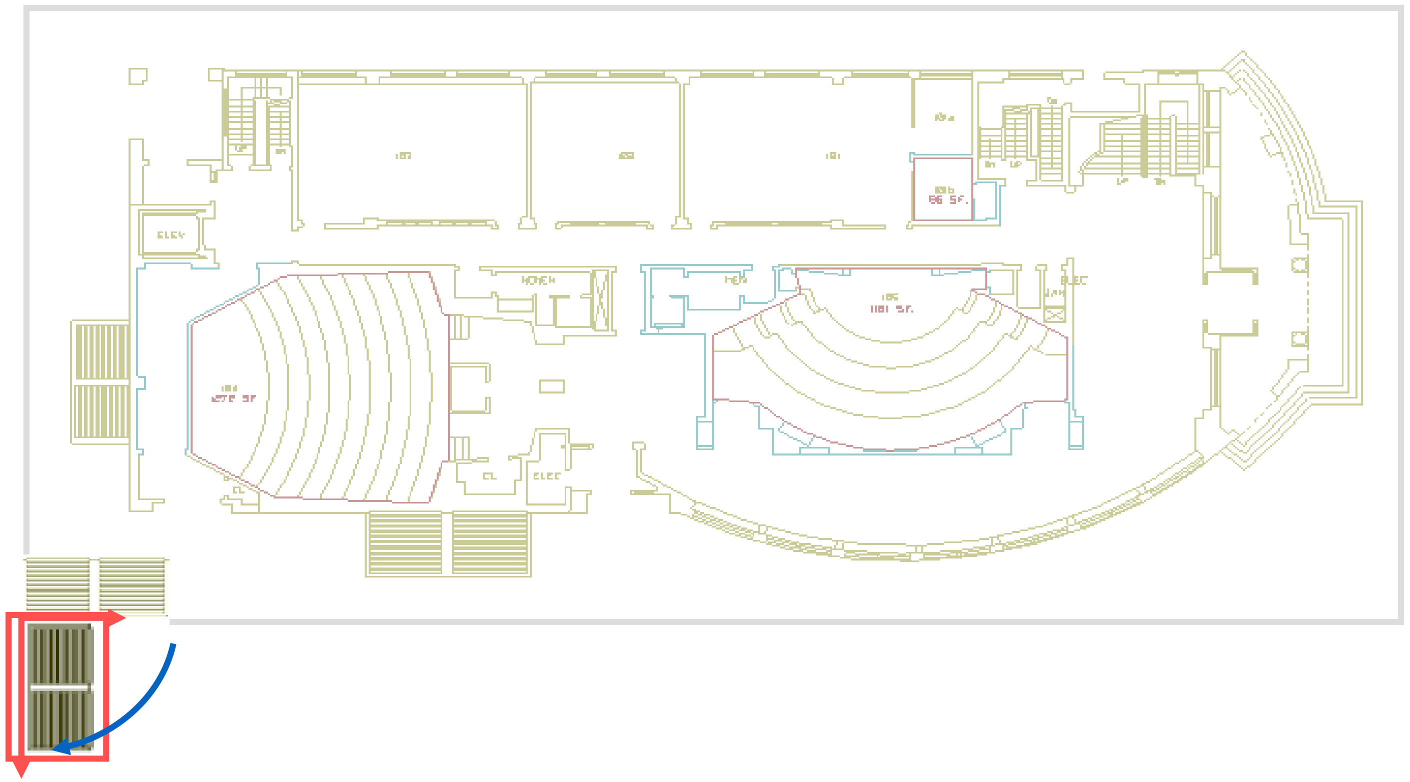


Object Transformations

Modeling
Coordinates

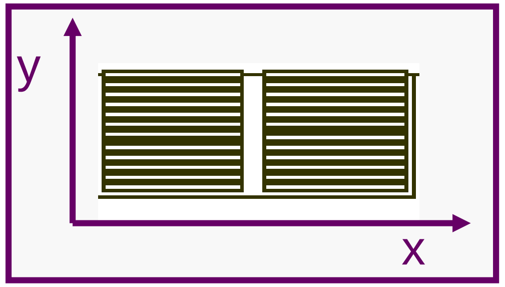


Scale .3, .3
Rotate -90

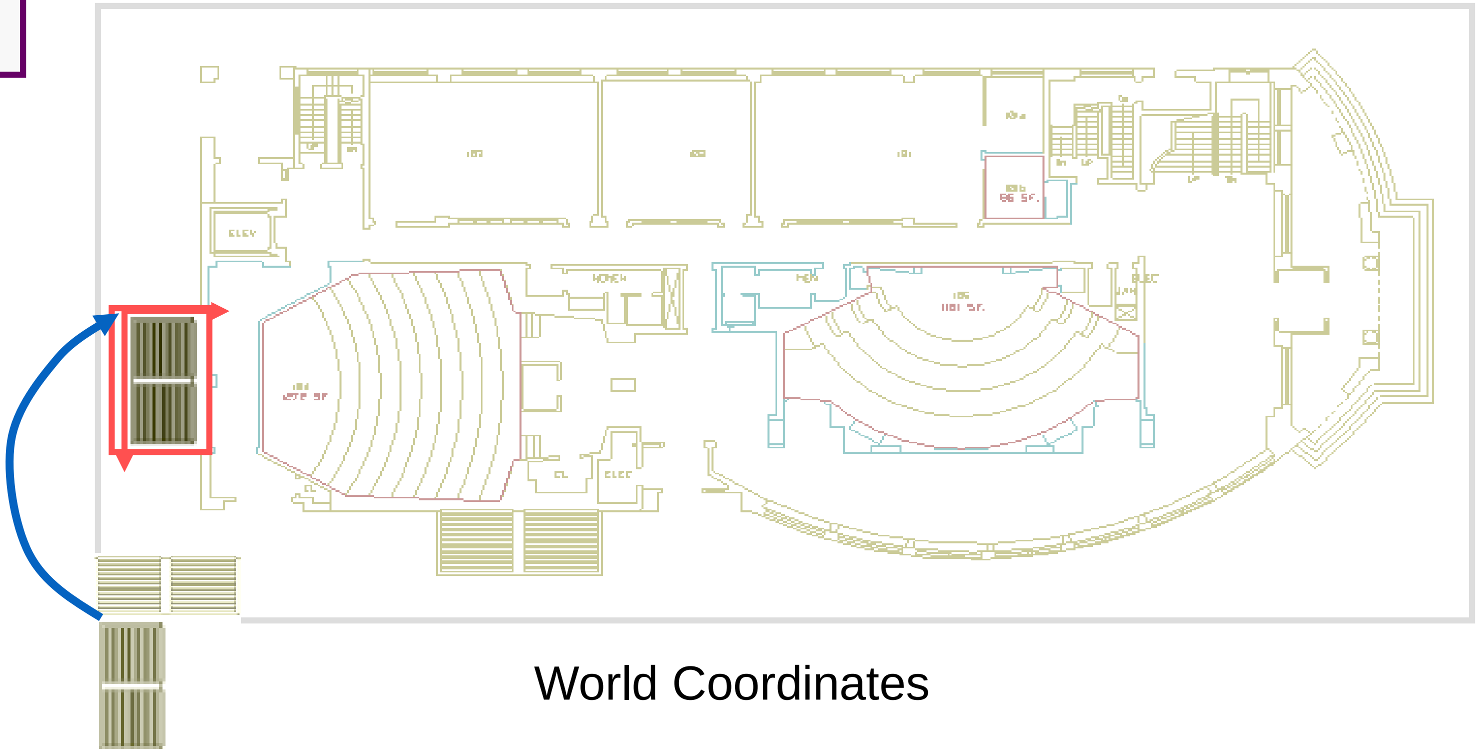


Object Transformations

Modeling Coordinates



Scale .3, .3
Rotate -90
Translate 5, 3



World Coordinates

Concepts Required 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, **w**)

You will need to understand these concepts!

Linear Algebra

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

Linear Algebra

- 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]**

Linear Algebra

Let’s calculate for each of the three stores.

- **Store A:**

$$\text{totalCost}_A = \sum_{i=1}^{4} P(A)_i q_i$$

= (0.20 • 6) + (0.93. • 5) + (0.64 • 1) + (1.20 • 2)
= (1.2 + 4.65 + 0.64 + 2.40)
= 8.89

- **Store B:**

$$\text{totalCost}_B = \sum^{4} P(B)_i q_i = 3.9 + 4.75 + 0.75 + 2.8$$

$$= 12.2$$

- **Store C:**

$$\text{totalCost}_C = \sum^{4} P(C)_i q_i = 5.7 + 5.5 + 0.9 + 7 = 19.1$$

Linear Algebra

- Can express these sums more compactly:

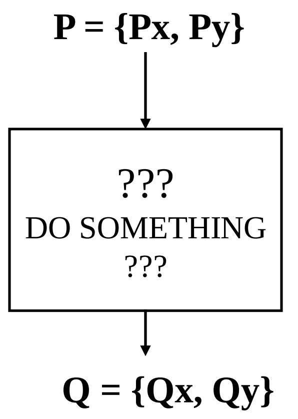
$$P(All) = \begin{bmatrix} totalCost_A \\ totalCost_B \\ totalCost_C \end{bmatrix} = \begin{bmatrix} 0.20 & 0.93 & 0.64 & 1.20 \\ 0.65 & 0.95 & 0.75 & 1.40 \\ 0.95 & 1.10 & 0.90 & 3.50 \end{bmatrix} \begin{bmatrix} 6 \\ 5 \\ 1 \\ 2 \end{bmatrix}$$

- 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

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

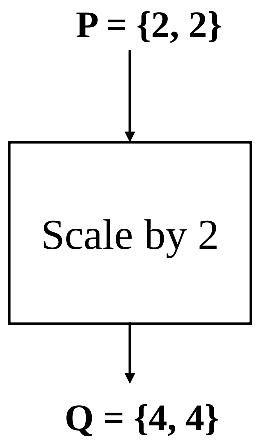
2D Transformations

- A 2D object transformation alters each point **P** into a new point **Q** using certain rules



- It therefore alters the co-ordinates of **P** {Px, Py} into new values which specify point **Q** {Qx,Qy}

- This can be expressed using some *function* **T**, that maps co-ordinate pairs into new co-ordinate pairs:
  - T ({Px,Py}) = {Qx,Qy}
  - May simply be denoted as T(P) = Q



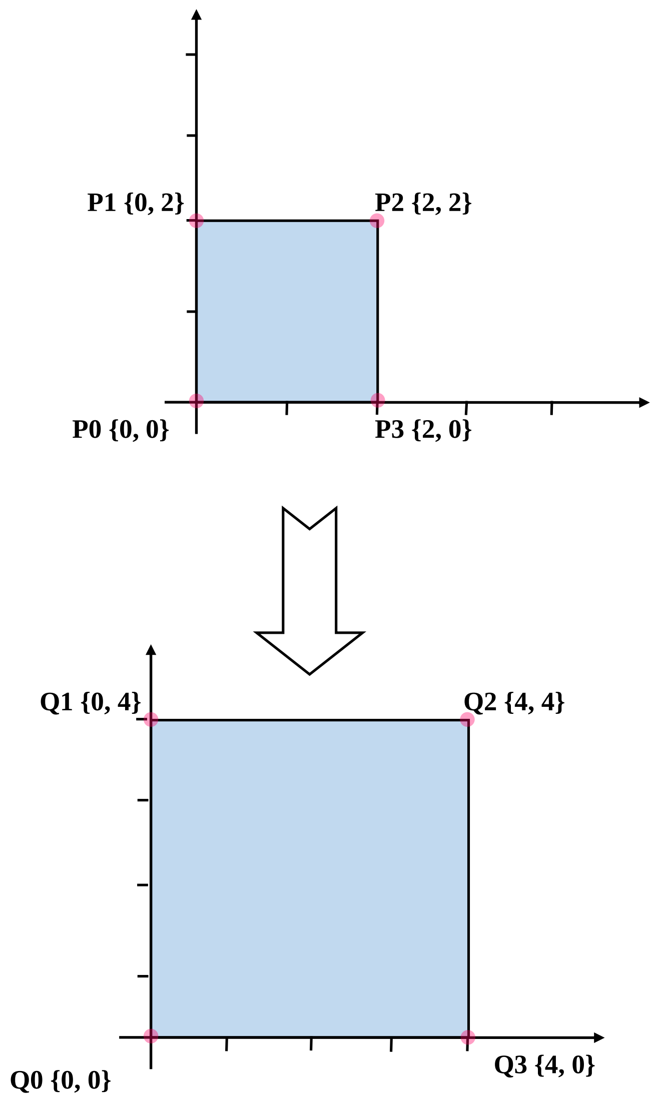
- e.g: P is the point {2, 2}, **T** is a transform that scales by a factor of 2. Then T(P) = T({2, 2}) = {4,4}

2D Transformations

- Whole collections of points may be transformed by the same transformation T, e.g. lines or circles

- The image of a line, L, under T, is the set of all *images* of the individual points of L.
- For most mappings of interest, this image is still a connected curve of some shape
- For some mappings, the result of transforming a line may no longer be a line

- **Affine Transformations**, however, do preserve lines, and are the most commonly-used transformations in computer graphics



Useful Properties of Affine Transformations

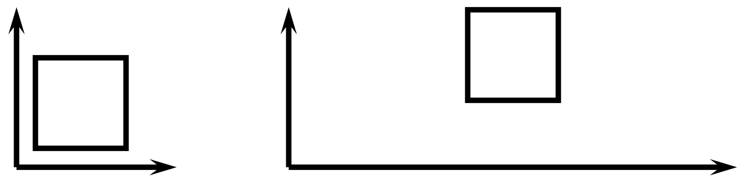
- **Preservation of lines (isometric)**
- **Preservation of parallelism (conformal)**
- **Preservation of proportional distances (rigid body)**

Why Useful?

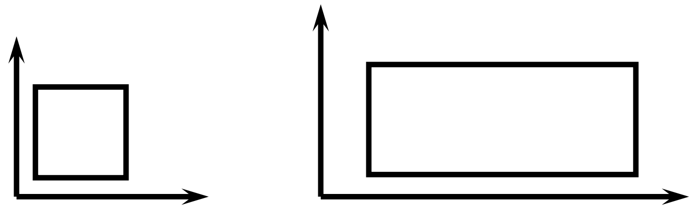
- They preserve lines, so the image of a straight line is another straight line.
- This vastly simplifies drawing transformed line segments.
- We need only compute the image of the two endpoints of the original line and then draw a straight line between them
- Preservation of co-linearity guarantees that polygons will transform into polygons
- Preservation of parallelism guarantees that parallelograms will transform into parallelograms
- Preservation of proportional distances means that mid-points of lines remain mid-points

Elementary Transformation

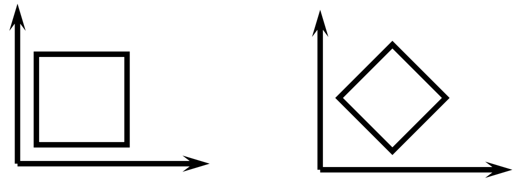
- Affine transformations are usually combinations of four elementary transformations:
- 1: Translation



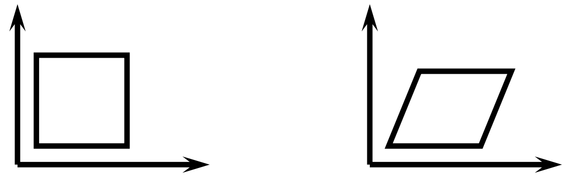
- 2: Scaling



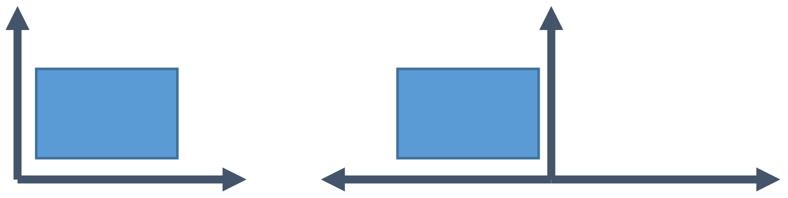
- 3: Rotation



- 4: Shearing

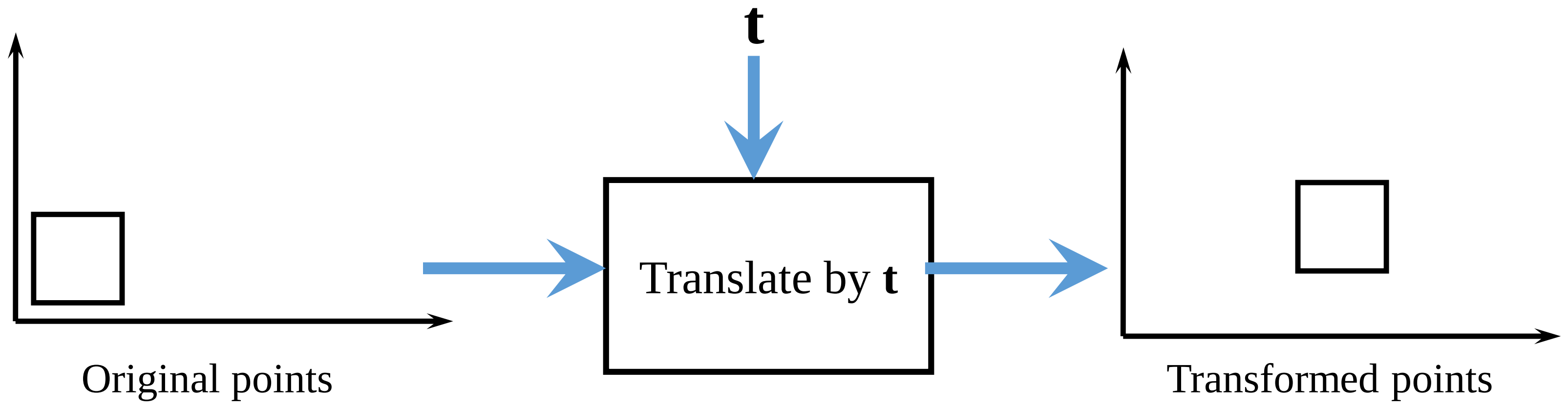


- 5: Reflection



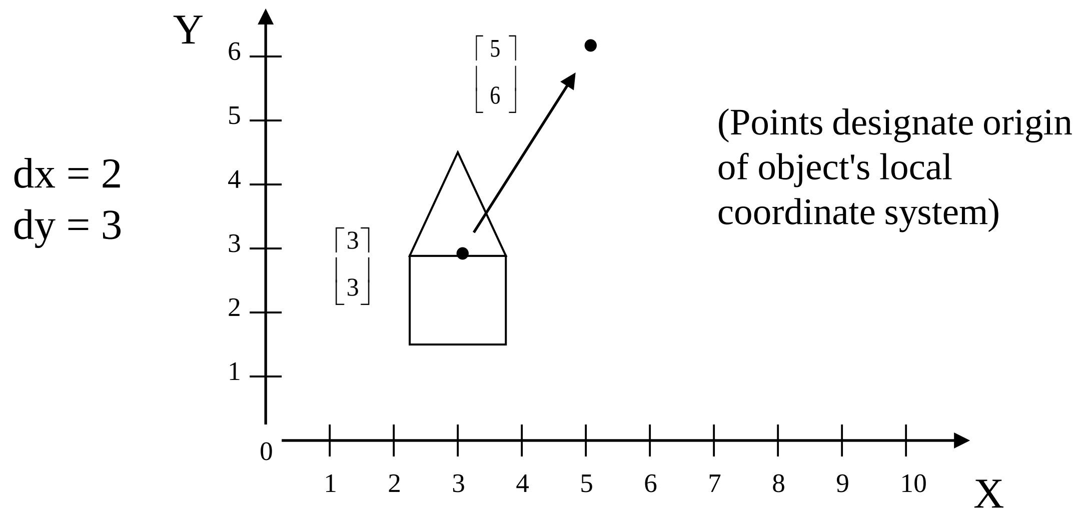
Translation

- A **translation** moves an object into a different position in a scene
- This is achieved by adding an offset/translation vector $\mathbf{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$
- In Vector notation: $\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$



Translation

Component-wise addition of vectors



dx = 2
dy = 3
Y
X
0
1
2
3
4
5
6
7
8
9
10
$\begin{bmatrix} 5 \\ 6 \end{bmatrix}$
$\begin{bmatrix} 3 \\ 3 \end{bmatrix}$
(Points designate origin of object's local coordinate system)

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

and $x' = x + dx$

$$y' = y + dy$$

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

Preserves lengths (isometric)

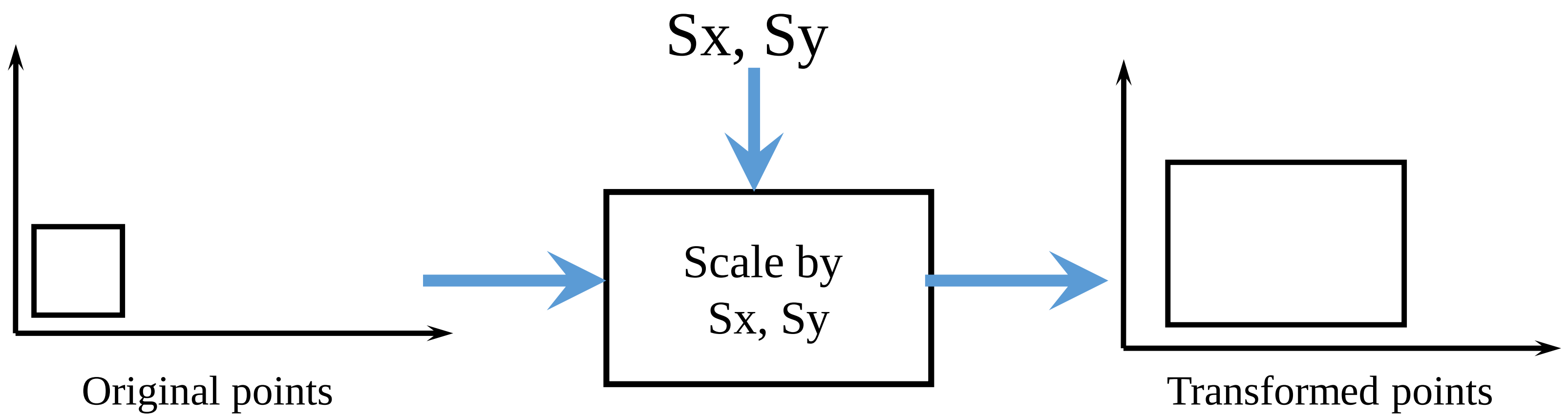
Preserves angles (conformal)

Translation is thus "rigid-body"

Scaling

- A scaling changes the size of an object with two scale factors, Sx and Sy (if Sx != Sy, then we have a non-uniform scale)

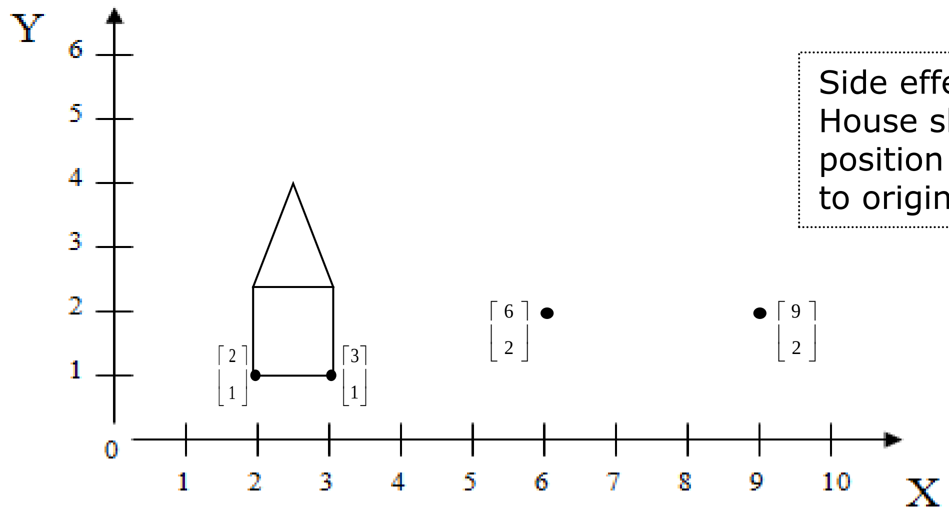
$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} P_x \times S_x \\ P_y \times S_y \end{bmatrix}$$



Scaling

$s_x = 3$

$s_y = 2$



Y
X
0
1
2
3
4
5
6
7
8
9
10
$\begin{bmatrix}2\\1\end{bmatrix}$
$\begin{bmatrix}3\\1\end{bmatrix}$
$\begin{bmatrix}6\\2\end{bmatrix}$
$\begin{bmatrix}9\\2\end{bmatrix}$

Side effect: House shifts position relative to origin

Component-wise scalar multiplication of vectors

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

and $$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \qquad \begin{aligned} x' &= s_x x \\ y' &= s_y y \end{aligned}$$

Does not preserve lengths

Does not preserve angles (except when scaling is uniform)

Scaling

Scaling factor s_x scales objects in the x direction, while s_y scales in the y direction. The transformation equations 5-10 can also be written in the matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \qquad (5\text{-}11)$$

or

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P} \qquad (5\text{-}12)$$

where **S** is the 2 by 2 scaling matrix in Eq. 5-11.

When s_x and s_y are assigned the same value, a **uniform scaling** is pro duced that maintains relative object proportions. Unequal values for s_x and s_y result in a **differential scaling** that is often used in design applications, where pictures are constructed from a few basic shapes that can be adjusted by scaling and positioning transformations

Scaling

We can control the location of a scaled object by choosing a position, called the **fixed point**, that is to remain unchanged after the scaling transformation. Coordinates for the fixed point (x_f, y_f) can be chosen as one of the vertices, the object centroid, or any other position (Fig. 5-8). A polygon is then scaled relative to the fixed point by scaling the distance from each vertex to the fixed point. For a vertex with coordinates (x, y), the scaled coordinates (x', y') are calculated as

$$x' = x_f + (x - x_f)s_x, \qquad y' = y_f + (y - y_f)s_y \tag{5-13}$$

We can rewrite these scaling transformations to separate the multiplicative and additive terms:

$$\begin{aligned} x' &= x \cdot s_x + x_f(1 - s_x) \\ y' &= y \cdot s_y + y_f(1 - s_y) \end{aligned} \tag{5-14}$$

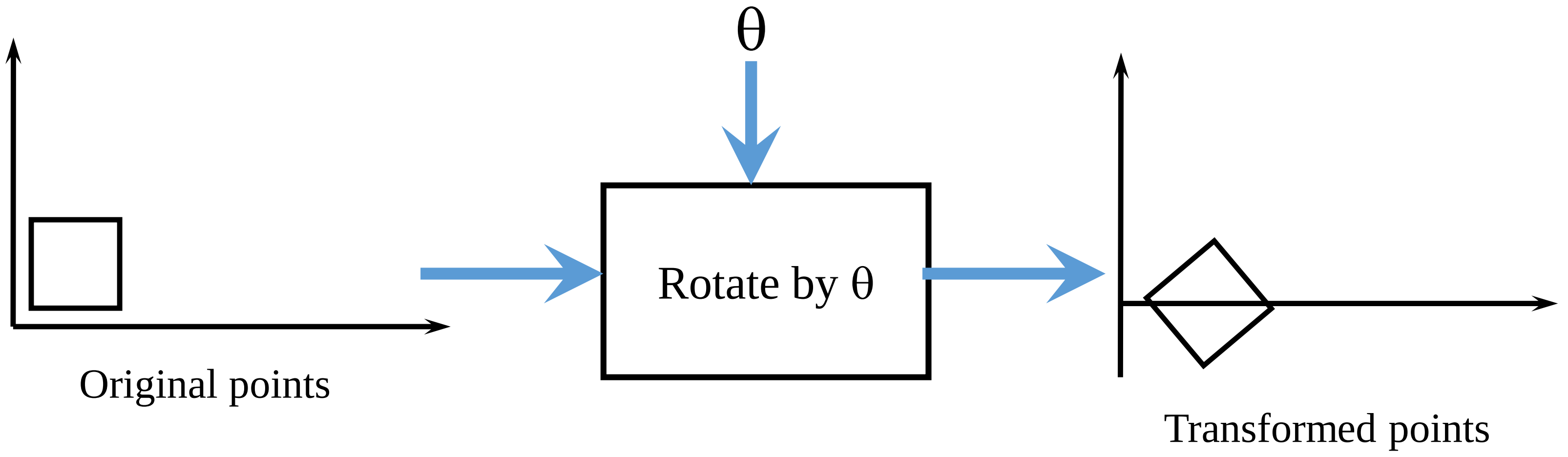
where the additive terms $x_f(1 - s_x)$ and $y_f(1 - s_y)$ are constant for all points in the object.

Rotation

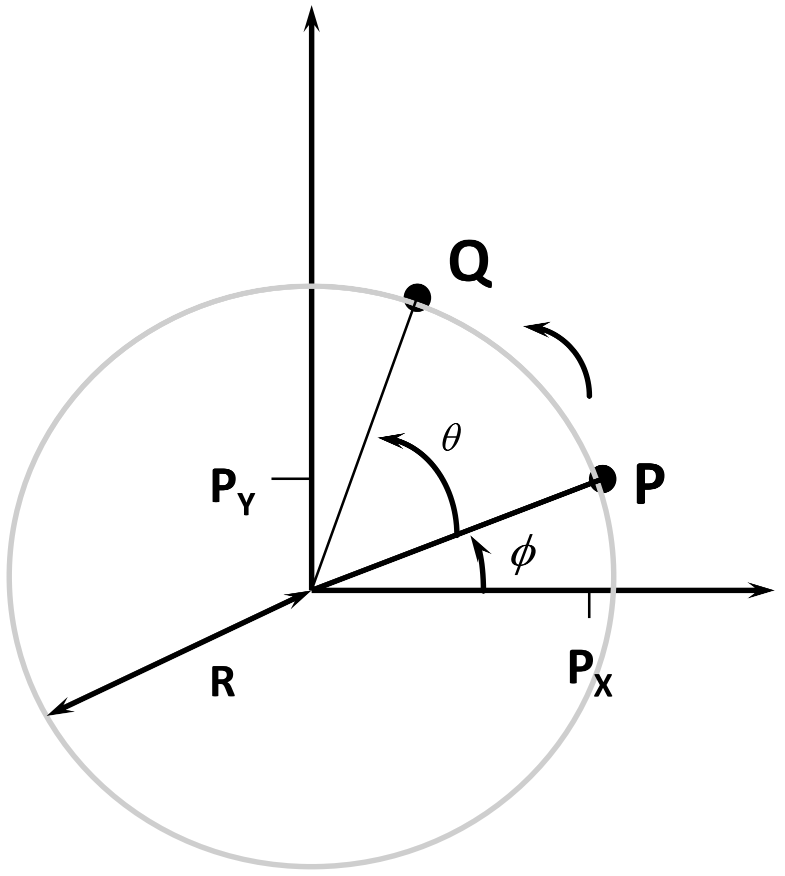
- Using the trigonometric relations, a point rotated by an angle θ about the origin is given by the following equations:

- So $Q_x = P_x \cos\theta - P_y \sin\theta \qquad Q_y = P_x \sin\theta + P_y \cos\theta$

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} P_x \cos\theta - P_y \sin\theta \\ P_x \sin\theta + P_y \cos\theta \end{bmatrix}$$



Rotation - derivation



Background info

$$Q_x = R\cos(\theta + \phi) \qquad [1]$$

$$Q_y = R\sin(\theta + \phi) \qquad [2]$$

$$P_x = R\cos(\phi) \qquad [3]$$

$$P_y = R\sin(\phi) \qquad [4]$$

$$\boxed{\begin{array}{l} \cos(\theta+\phi) = \cos(\theta)\cos(\phi) - \sin(\theta)\sin(\phi) \\ \sin(\theta+\phi) = \sin(\theta)\cos(\phi) + \cos(\theta)\sin(\phi) \end{array}}$$

$$[1] \Rightarrow Q_x = R\cos(\theta)\cos(\phi) - R\sin(\theta)\sin(\phi)$$

Substituting from [3] and [4]…

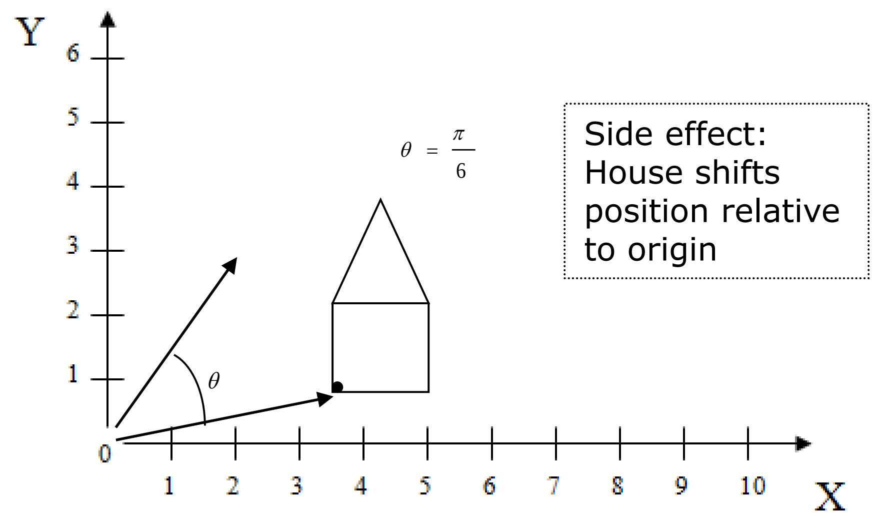
$$Q_x = P_x\cos(\theta) - P_y\sin(\theta)$$

Similarly from [2]…

$$Q_y = P_y\cos(\theta) + P_x\sin(\theta)$$

Rotation

Rotate by θ about the origin



Side effect: House shifts position relative to origin

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

and $x' = x \cos \Theta - y \sin \Theta$

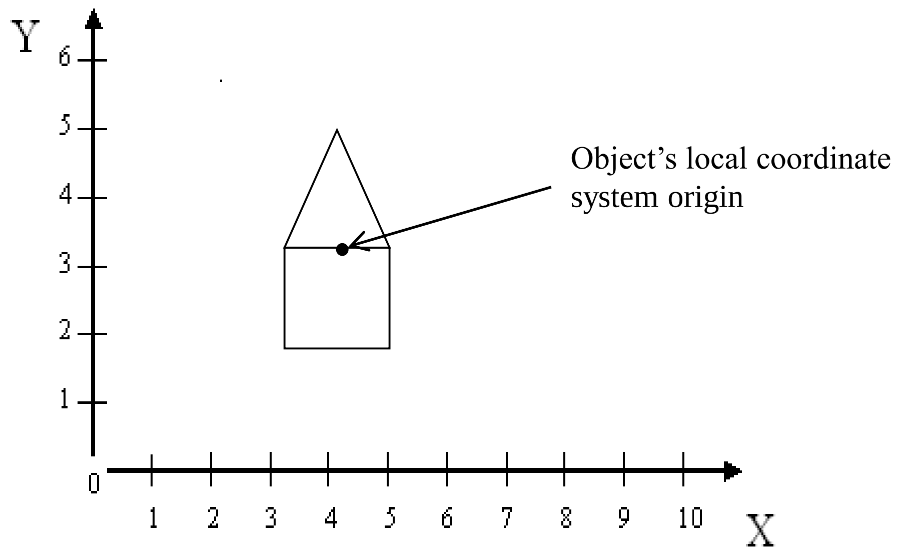
$y' = x \sin \Theta + y \cos \Theta$

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

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

Rotation



- 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...

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$

$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$$

Rotation

With the column-vector representations 5-2 for coordinate positions, we can write the rotation equations in the matrix form:

$$\mathbf{P}' = \mathbf{R} \cdot \mathbf{P} \tag{5-7}$$

where the rotation matrix is

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5-8}$$

When coordinate positions are represented as row vectors instead of column vectors, the matrix product in rotation equation 5-7 is transposed so that the transformed row coordinate vector $[x' \; y']$ is calculated as

$$\begin{aligned} \mathbf{P}'^T &= (\mathbf{R} \cdot \mathbf{P})^T \\ &= \mathbf{P}^T \cdot \mathbf{R}^T \end{aligned}$$

where $\mathbf{P}^T = [x \; y]$, and the transpose $\mathbf{R}^T$ of matrix $\mathbf{R}$ is obtained by interchanging rows and columns. For a rotation matrix, the transpose is obtained by simply changing the sign of the sine terms.

Shearing

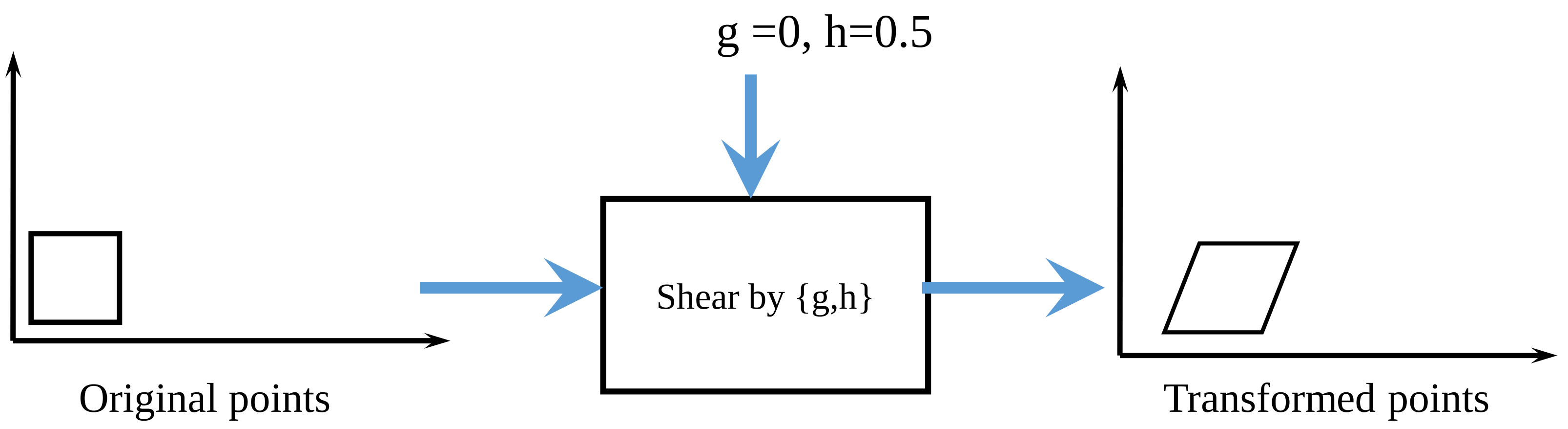
- A shearing affects an object in a particular direction (in 2D, it’s either in the **x** or in the **y** direction)
- A shear in the **x** direction would be as follows:

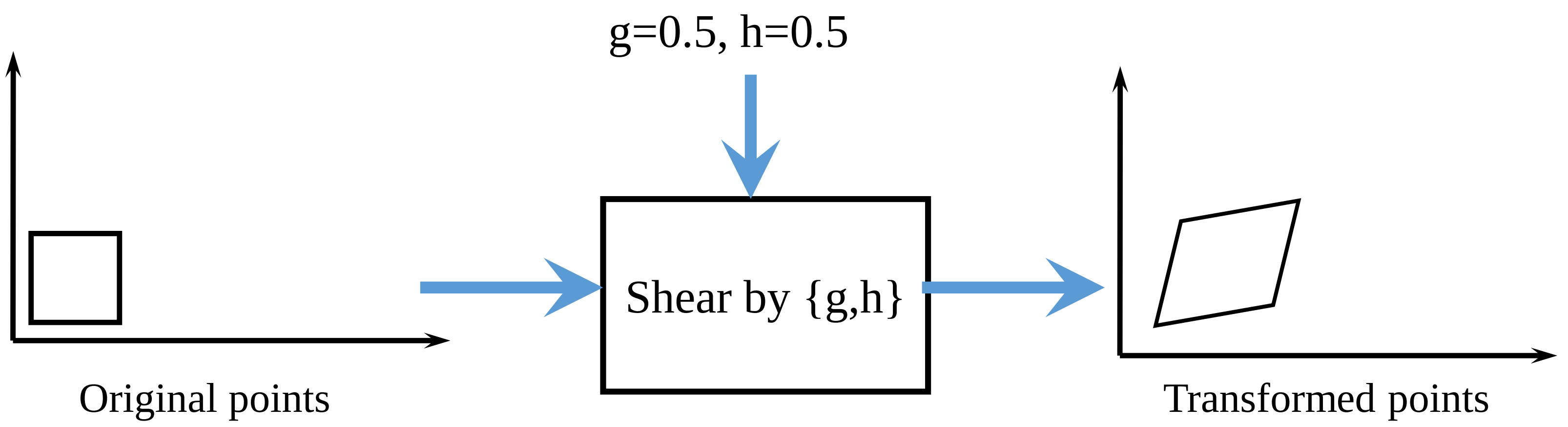
$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} P_x + hP_y \\ P_y \end{bmatrix}$$

- The quantity h specifies what fraction of the y-coordinate should be added to the x-coordinate, and may be positive or negative
- More generally: a simultaneous shear in both the x and y directions would be

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} P_x + hP_y \\ P_y + gP_x \end{bmatrix}$$

Shearing


g =0, h=0.5
Shear by {g,h}
Original points
Transformed points

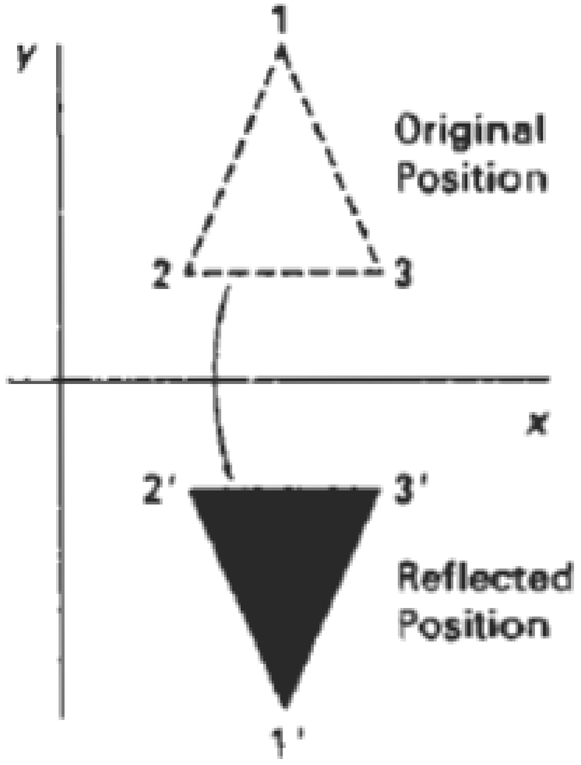

g=0.5, h=0.5
Shear by {g,h}
Original points
Transformed points

Reflection

A **reflection** is a transformation that produces a mirror image of an object. The mirror image for a two-dimensional reflection is generated relative to an **axis of reflection** by rotating the object 180° about the reflection axis. We can choose an axis of reflection in the *xy* plane or perpendicular to the *xy* plane. When the reflection axis is a line in the *xy* plane, the rotation path about this axis is in a plane perpendicular to the *xy* plane. For reflection axes that are perpendicular to the *xy* plane, the rotation path is in the *xy* plane. Following are examples of some common reflections.

Reflection about the line $y = 0$, the x axis, is accomplished with the transformation matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}48)$$

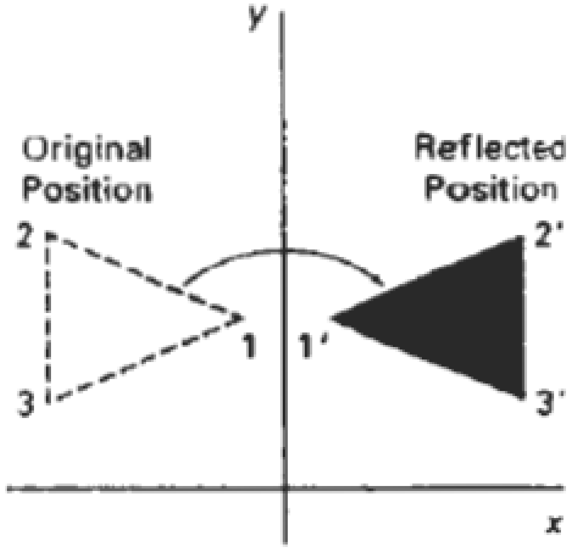


Reflection

A reflection about the y axis flips x coordinates while keeping y coordinates the same. The matrix for this transformation is

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}49)$$

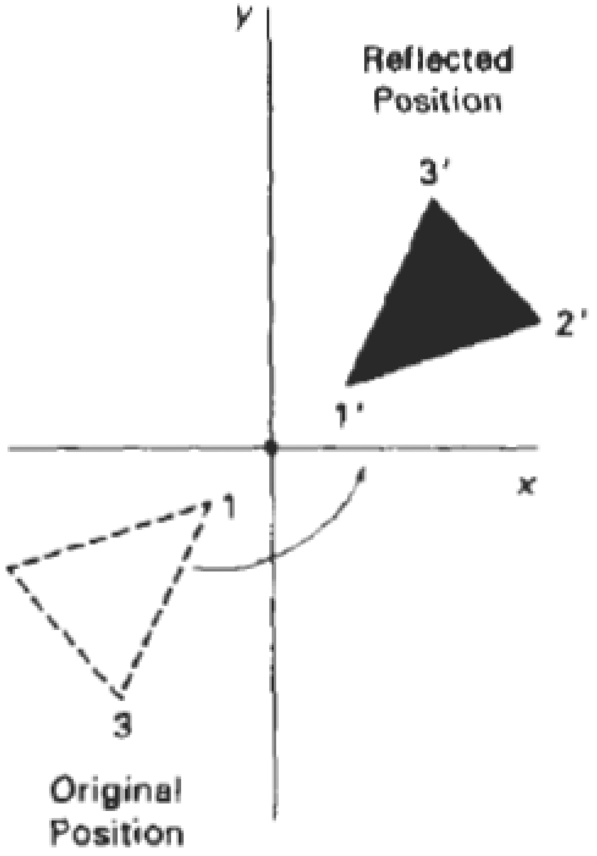
Figure 5-17 illustrates the change in position of an object that has been reflected about the line $x = 0$. The equivalent rotation in this case is 180° through three-dimensional space about the y axis.



Reflection

We flip both the *x* and *y* coordinates of a point by reflecting relative to an axis that is perpendicular to the *xy* plane and that passes through the coordinate origin. This transformation, referred to as a reflection relative to the coordinate origin, has the matrix representation:

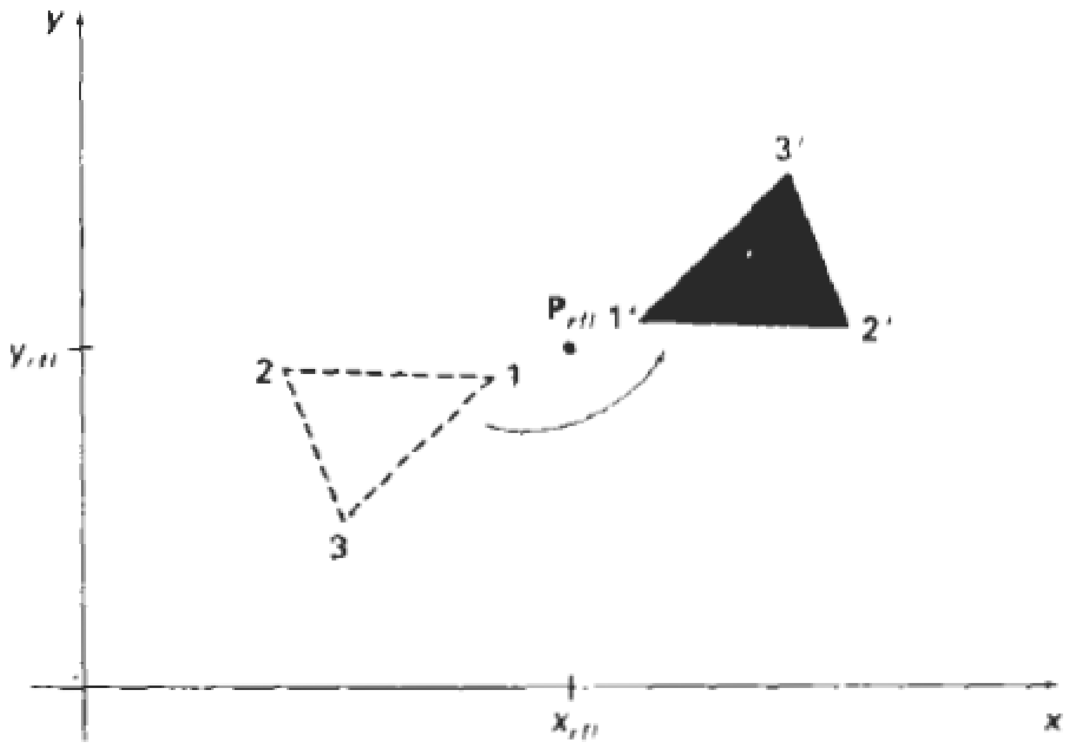
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}50)$$



Reflection

An example of reflection about the origin is shown in Fig. 5-18. The reflection matrix 5-50 is the rotation matrix $\mathbf{R}(\theta)$ with $\theta = 180°$. We are simply rotating the object in the *xy* plane half a revolution about the origin.

Reflection 5-50 can be generalized to any reflection point in the *xy* plane (Fig. 5-19). This reflection is the same as a 180° rotation in the *xy* plane using the reflection point as the pivot point.

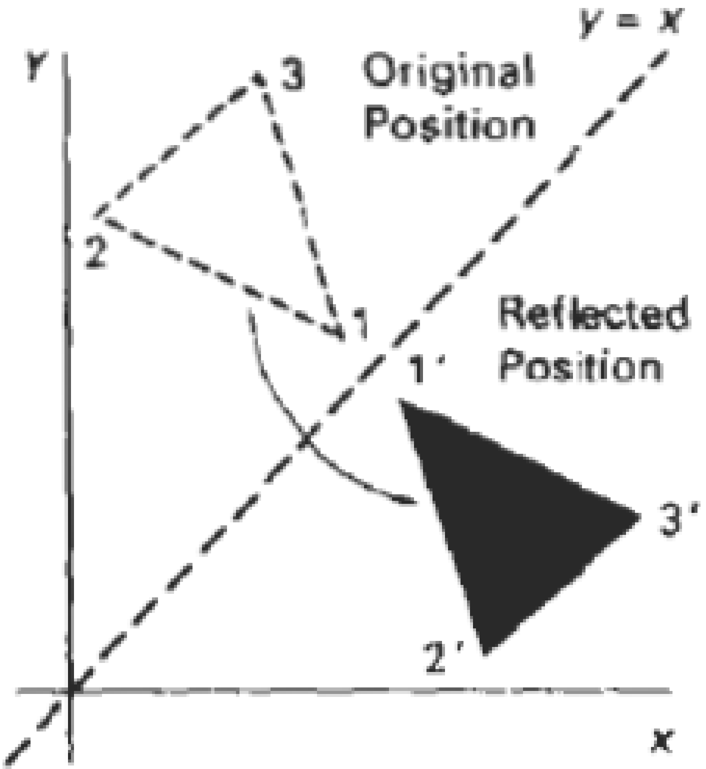


Reflection

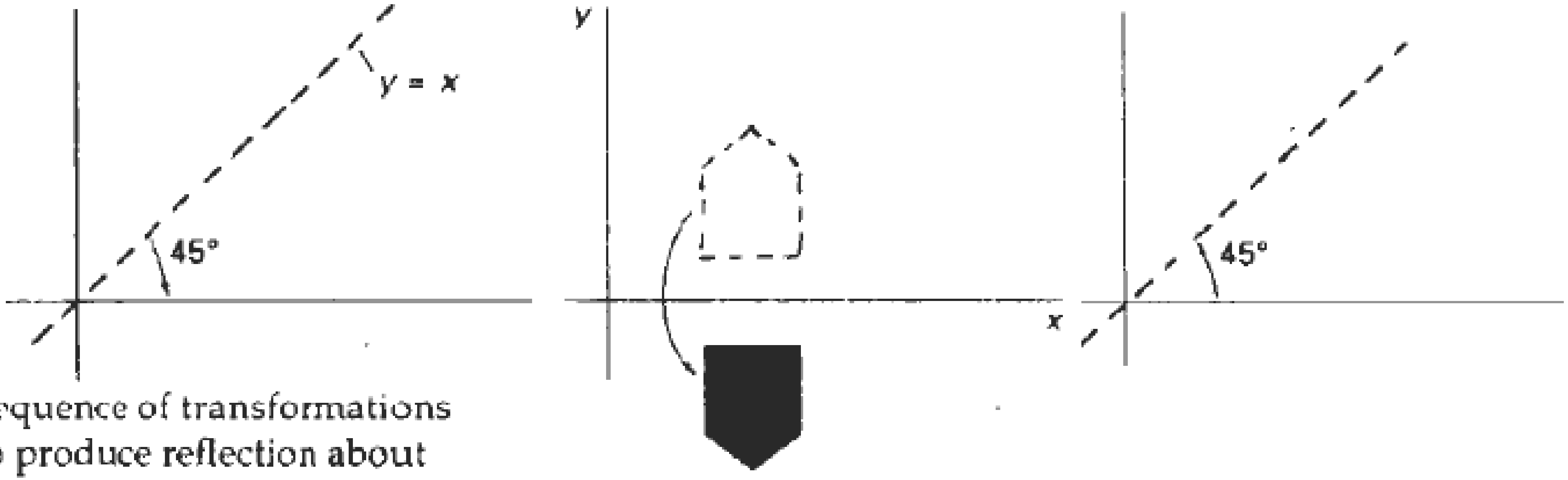
If we chose the reflection axis as the diagonal line $y = x$ (Fig. 5-20), the reflection matrix is

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}51)$$

We can derive this matrix by concatenating a sequence of rotation and coordinate-axis reflection matrices. One possible sequence is shown in Fig. 5-21. Here, we first perform a clockwise rotation through a 45° angle, which rotates the line $y = x$ onto the x axis. Next, we perform a reflection with respect to the x axis. The final step is to rotate the line $y = x$ back to its original position with a counterclockwise rotation through 45°. An equivalent sequence of transformations is first to reflect the object about the x axis, and then to rotate counterclockwise 90°.



Reflection

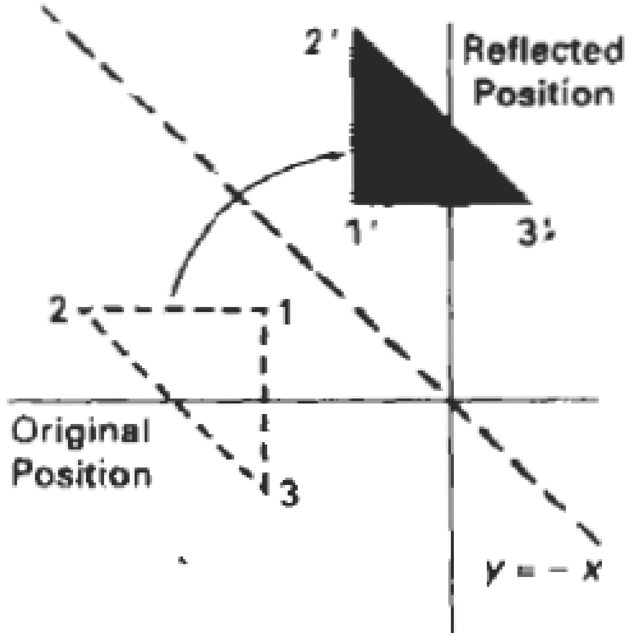


Sequence of transformations to produce reflection about the line $y = x$: (a) clockwise rotation of 45°, (b) reflection about the x axis, and (c) counterclockwise rotation by 45°.

Reflection

To obtain a transformation matrix for reflection about the diagonal $y = -x$, we could concatenate matrices for the transformation sequence: (1) clockwise rotation by 45°, (2) reflection about the y axis, and (3) counterclockwise rotation by 45°. The resulting transformation matrix is

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}52)$$



Reflection

Reflections about any line $y = mx + b$ in the xy plane can be accomplished with a combination of translate-rotate-reflect transformations. In general, we first translate the line so that it passes through the origin. Then we can rotate the line onto one of the coordinate axes and reflect about that axis. Finally, we restore the line to its original position with the inverse rotation and translation transformations.

We can implement reflections with respect to the coordinate axes or coordinate origin as scaling transformations with negative scaling factors. Also, elements of the reflection matrix can be set to values other than ± 1. Values whose magnitudes are greater than 1 shift the mirror image farther from the reflection axis, and values with magnitudes less than 1 bring the mirror image closer to the reflection axis.

Transformations Summary

- 1: Translation

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- 2: Scaling

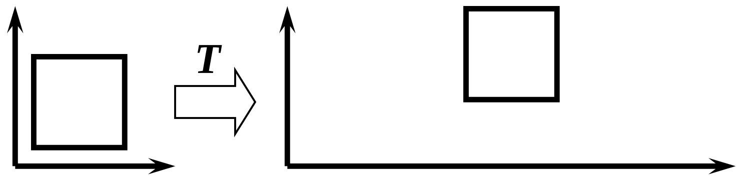
$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

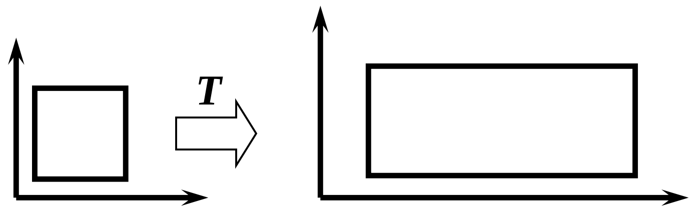
- 3: Rotation

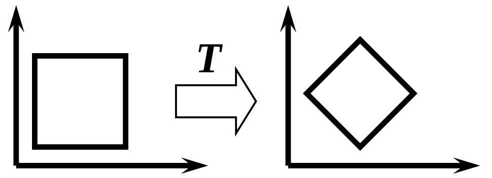
$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

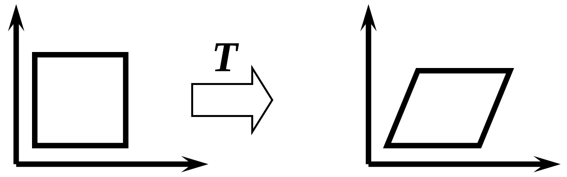
- 4: Shearing

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} 1 & h \\ g & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$









Matrix Representation

- All affine transformations in 2D can be generically described in terms of a generic equation as follows:

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- i.e. $\mathbf{Q} = \mathbf{MP} + \mathbf{t}$

Problem

- An affine transformation is composed of a linear combination followed by a translation
- Unfortunately, the translation portion is not a matrix multiplication but must instead be added as an extra term, or vector – this is inconvenient

Homogeneous Coordinates

- The “trick” we use is to add an additional component 1 to both P and Q, and also a third row and column to M, consisting of zeros and a 1
- i.e.

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Translation by {tx, ty}

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

Rotate by θ:

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

Scale by Sx, Sy

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

Shear by g, h:

$$M = \begin{bmatrix} 1 & h & 0 \\ g & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Homogeneous Coordinates

- Translation, scaling and rotation are expressed as:

translation:	$v' = v + t$
scale:	$v' = Sv$
rotation:	$v' = Rv$

- 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

Multiple Transformations

- It is rare that we want to perform just one elementary transformation.
- Usually an application requires that we build a complex transformation out of several elementary ones
  - e.g. translate an object, rotate it, and scale it, all in one move

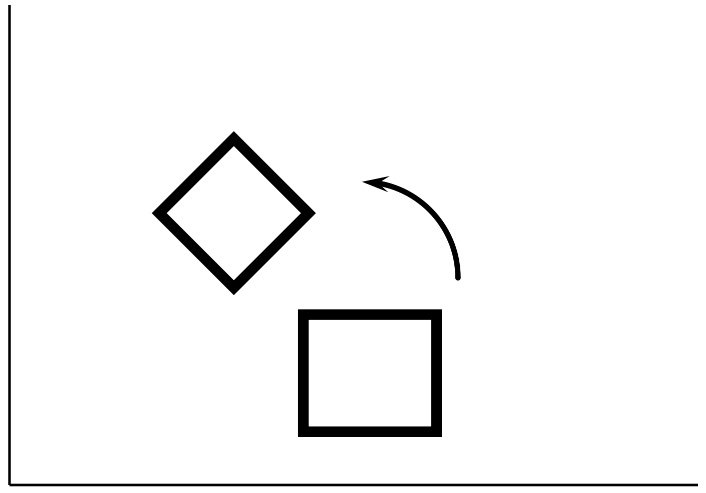
- These individual transformations combine into one overall transformation

- This is called the composition of transformations.
- The composition of two or more affine transformations is also an affine transformation

Rotation and Scaling

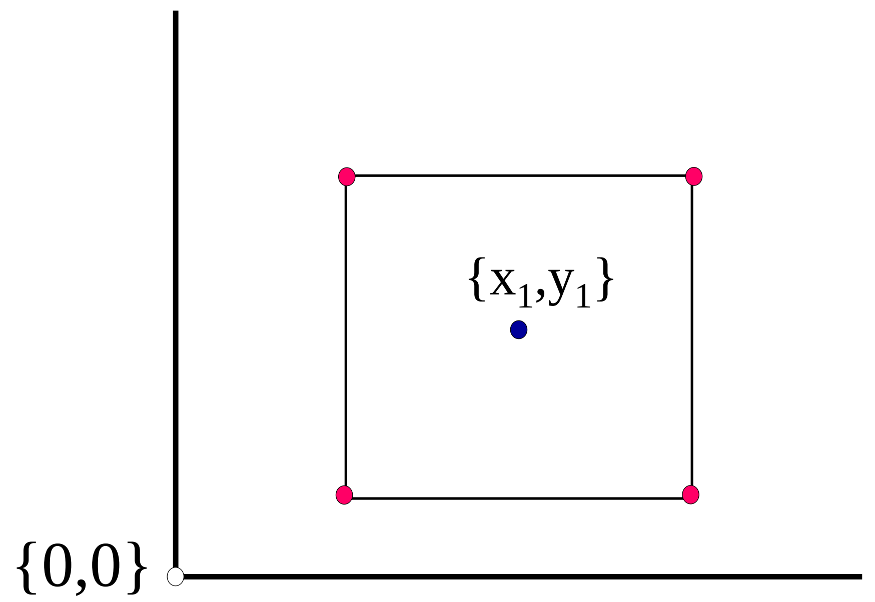
- The simple versions of rotation and scaling have been based around the origin.
- This means that when we rotate or scale, the object will also move, with respect to the origin

Not only is the object rotated, but it also moves around the origin



Pivotal points

- Often we wish to rotate or scale with respect to some pivotal point, not the origin
- Most significantly, we often wish to rotate or scale an object about its centre, or midpoint
- In this way, the object's location does not change
- To do this, we relate the rotation or scaling about the pivotal point V, to an elementary rotation or scaling about the origin
    - We first translate all points so that V coincides with the origin
    - We then rotate or about the origin
    - then all points are translated back, so that V is restored to its original location



To rotate the square by angle: θ about the point $\{x_1, y_1\}$

- Translate all points through $\{-x_1,-y_1\}$
- Rotate all points about the origin by θ
- Translate all points back through $\{x_1y_2\}$

Rotation about a pivot point

$$\begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_1(1-\cos\theta) + y_1\sin\theta \\ \sin\theta & \cos\theta & y_1(1-\cos\theta) - x_1\sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

Why composition?

- One of the main reasons for composing transformations is computational efficiency

  - Suppose you want to apply two affine transformations to the N vertices of a polygonal object
  - It requires 18 multiplications* to compose the transformations, and 6 multiplications to apply a transformation to a point
  - So, "compose-then-apply" requires 6N+18 multiplications

$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & t_{x1} \\ c_1 & d_1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} a_2 & b_2 & t_{x2} \\ c_2 & d_2 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix}$$

$$and \quad \begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

  - 12N multiplications are required if each transformation is applied separately (and possibly

$$\begin{bmatrix} P'_x \\ P'_y \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & t_{x1} \\ c_1 & d_1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} a_2 & b_2 & t_{x2} \\ c_2 & d_2 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} P'_x \\ P'_y \\ 1 \end{bmatrix}$$

  - For models with any significant number of vertices {which will be most}, 6N+18 is much less than 12N
    - e.g. if N=10000,   6N+18 = 60018, 12N = 12000
    - i.e. about half the effort

* : clever graphics systems will take into account that the bottom row doesn't need to be multiplied out. But even in this case Compositions takes about half the time (9N+27   vs.   18N)

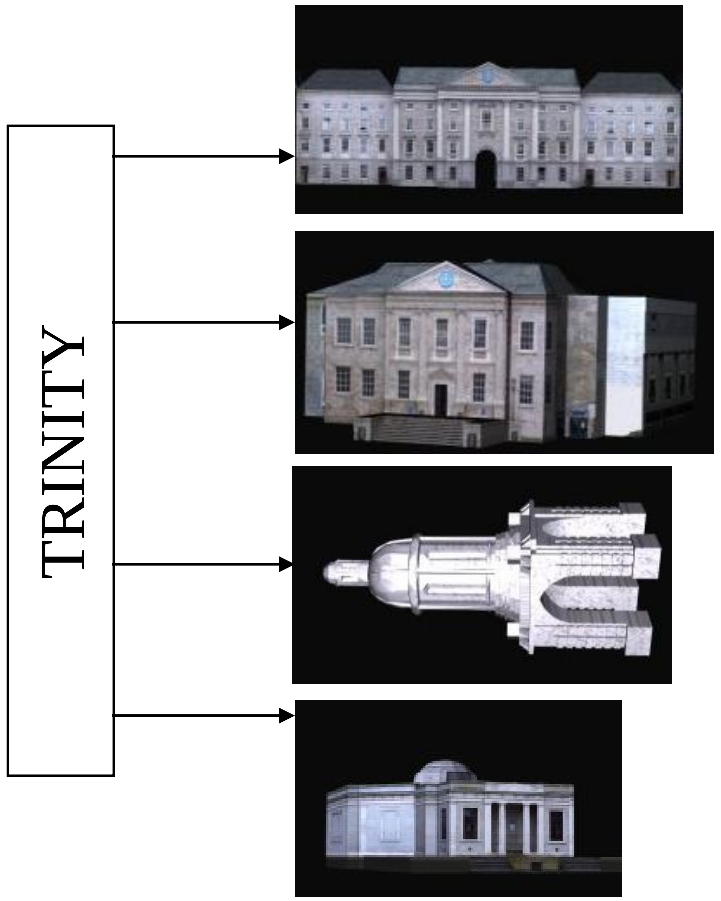
Local vs. Global Co-ordinates

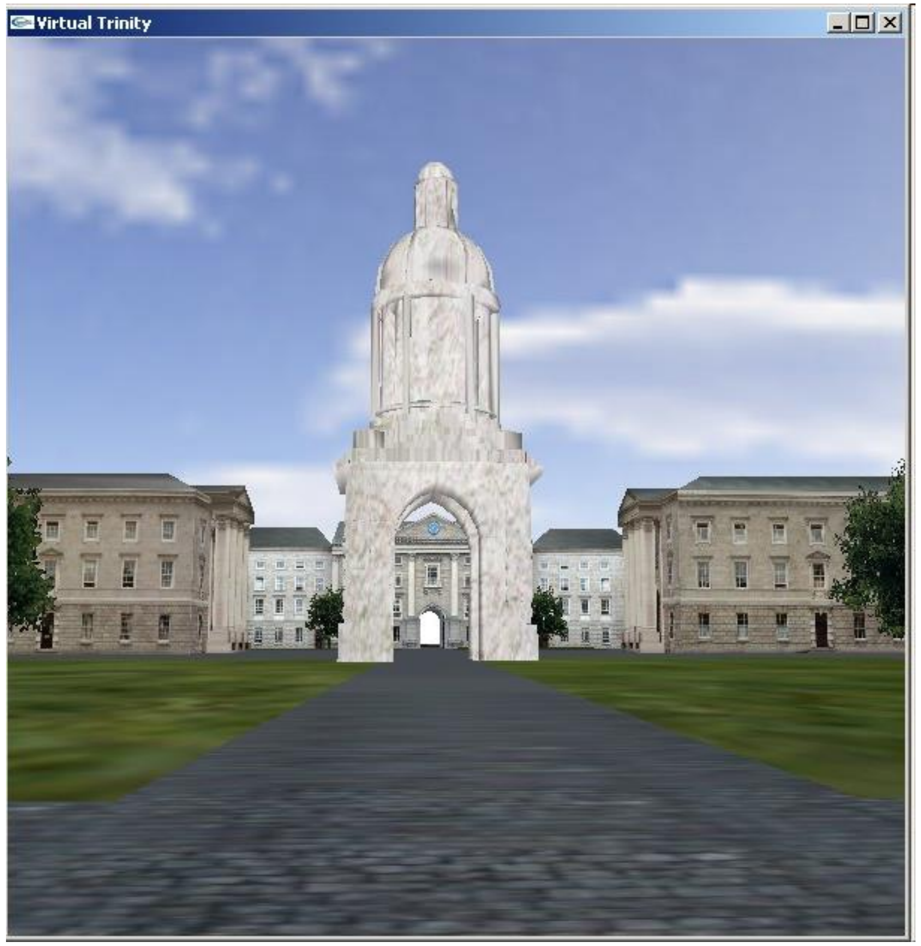
- In practice we find that it is cumbersome in most applications to have to recalculate the individual points that make up an object
- It is more common to define the object with reference to some *local* co-ordinate origin (or reference point) and axis.
- We then specify through a transformation matrix how this reference frame relates to the Global reference frame.

Modelling Scenes

Modelling complex scenes made up many objects is a two step process of modelling individual objects and then placing these in the scene (using **transforms**)

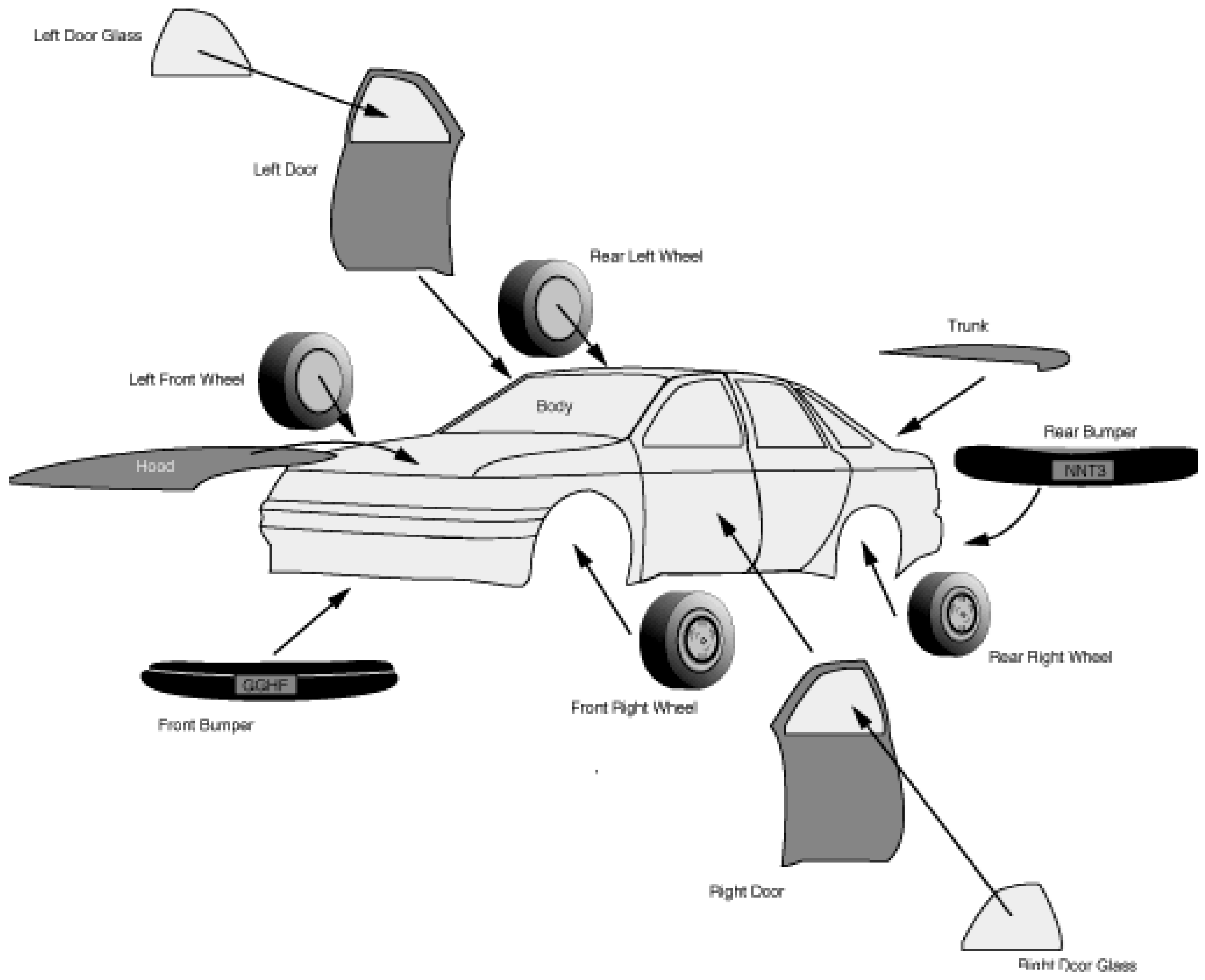
Scene Graphs





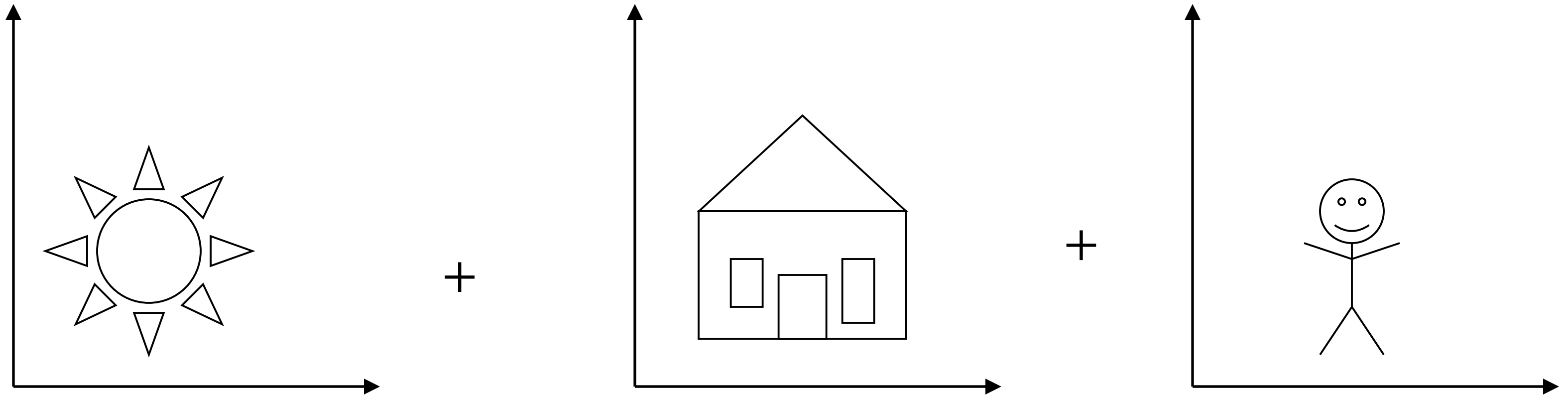
Individual object geometries are first modelled. These are then linked using positional/orientational relationships (transforms)

A Scene Graph



Body
- **Hood**
- **Trunk**
- **Front Bumper**
- **Rear Bumper**
- **Left Door**
  - **Left Door Glass**
- **Right Door**
  - **Right Door Glass**
- **Left Front Wheel**
- **Right Front Wheel**
- **Left Rear Wheel**
- **Right Rear Wheel**

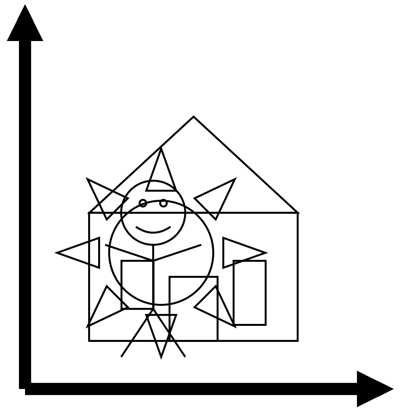
There can be multiple levels of transforms e.g. the “door glass” is linked to the “door” which is linked to the “body”

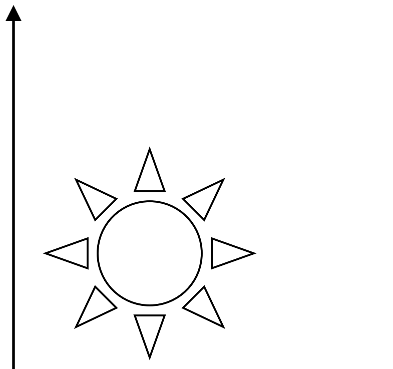
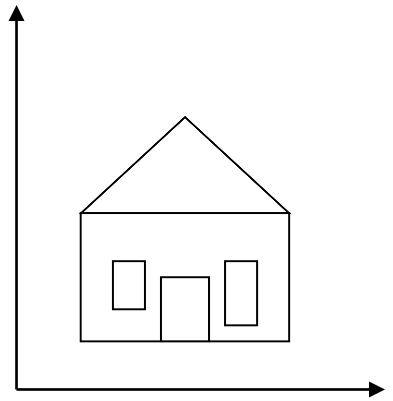
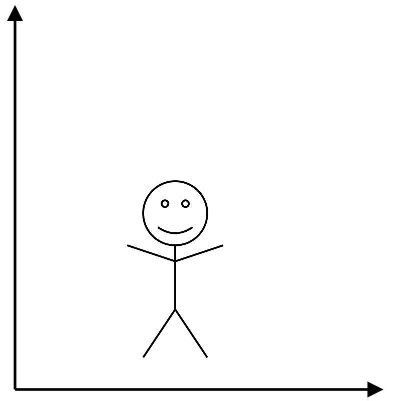


Usually we model individual objects based on a local coordinate system

This of course shouldn’t mean all objects need to share the same transformations

=



+

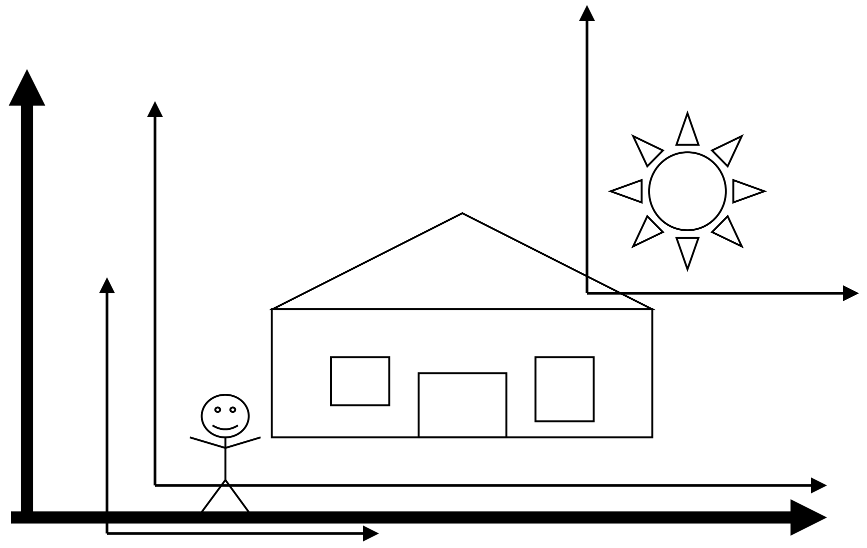
+
transform

+
transform

+
transform

Obviously we want something more versatile

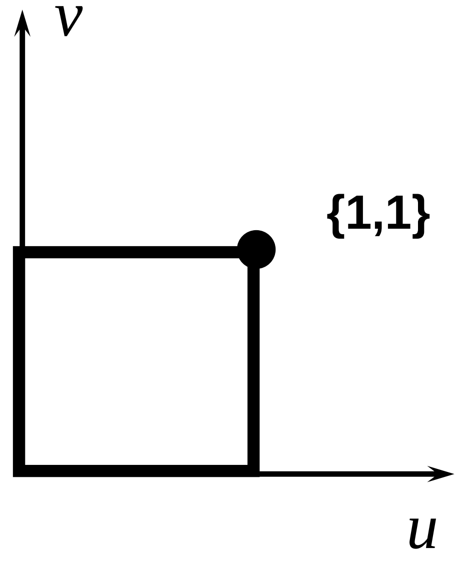
=



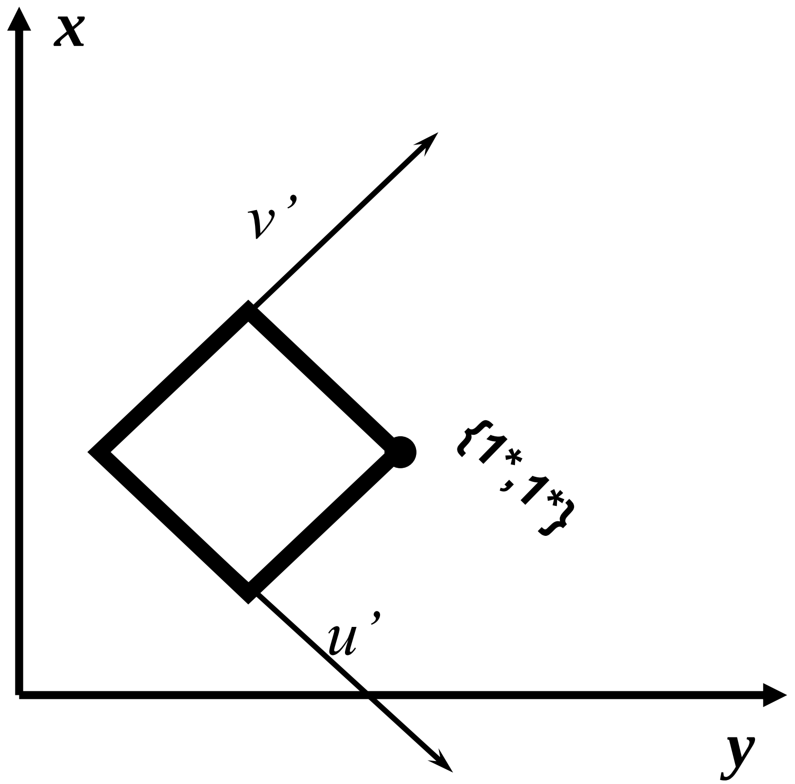
Current Transformation Matrix

- One way to do this is by using a single object to represent the CURRENT TRANSFORMATION MATRIX {CTM}
- All transformation functions affect only the CTM
- When a draw procedure is finally called it draws based on the CTM
- The CTM might be seen as representing the “current coordinate frame” for drawing

Coordinate Transforms

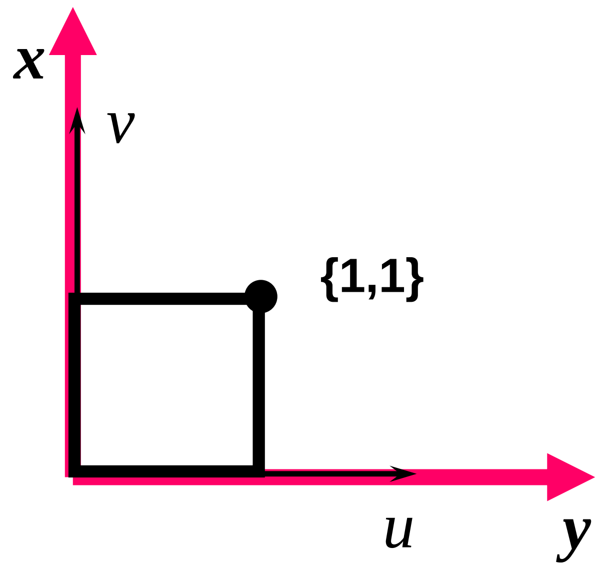


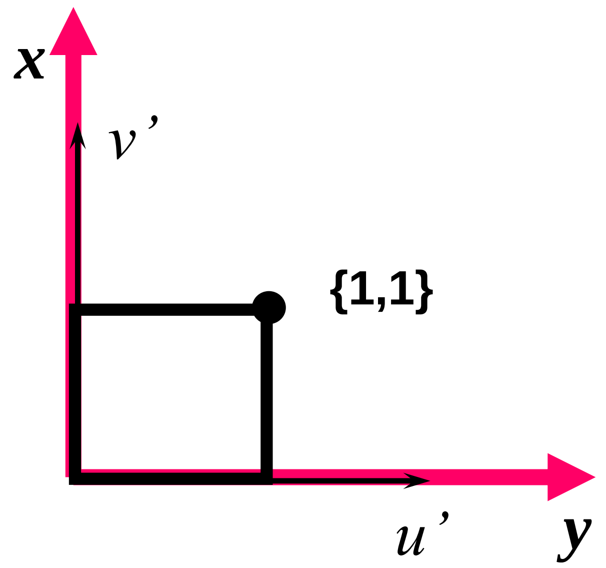
Object defined in **Local Coordinate System**



Object after transformation in **Global Coordinate System**

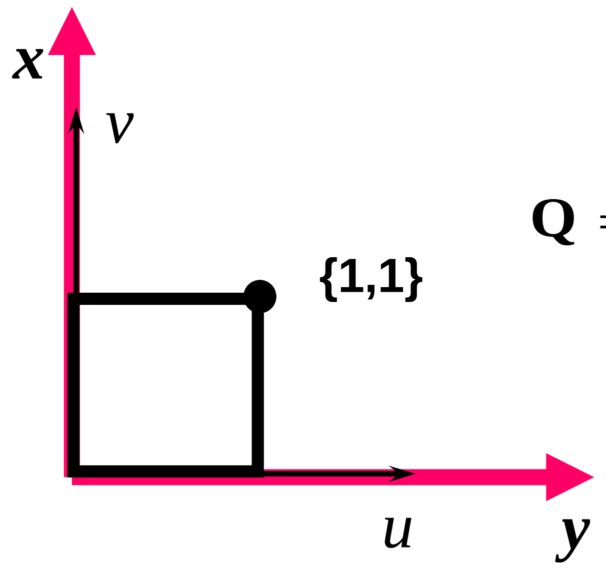
Identity



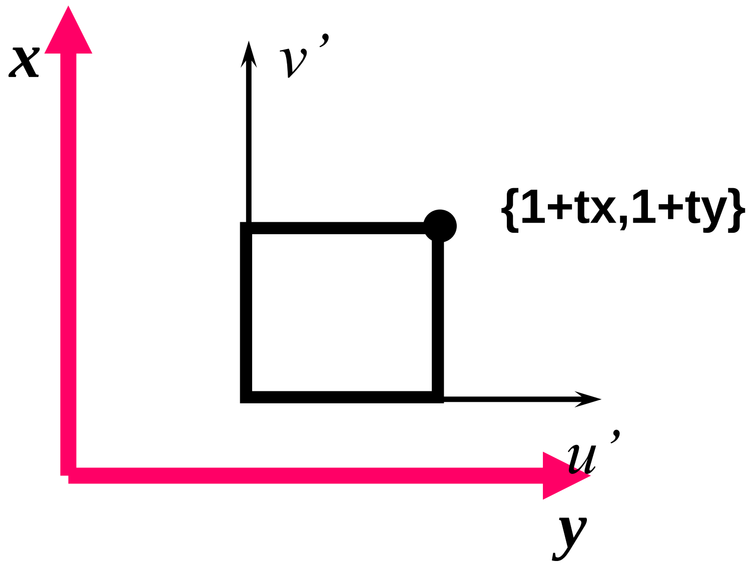


$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{P}$$

Translation



$$\mathbf{Q} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & tx \\ \mathbf{0} & \mathbf{1} & ty \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \mathbf{P}$$



$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & tx \\ \mathbf{0} & \mathbf{1} & ty \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} tx \\ ty \\ 1 \end{bmatrix}$$

origin

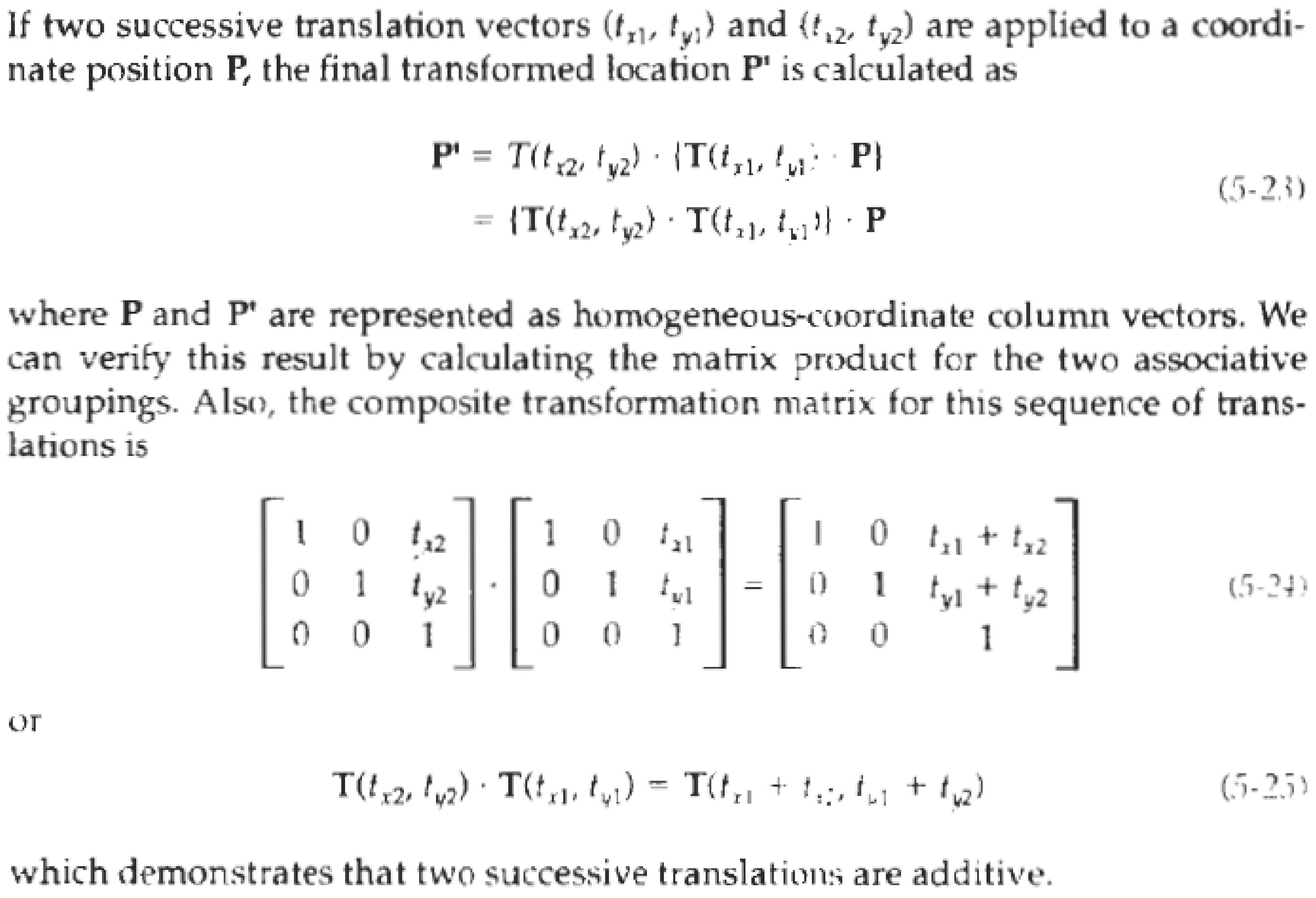
$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & tx \\ \mathbf{0} & \mathbf{1} & ty \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1+tx \\ ty \\ 1 \end{bmatrix}$$

v

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & tx \\ \mathbf{0} & \mathbf{1} & ty \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} tx \\ 1+ty \\ 1 \end{bmatrix}$$

u {0, 1, 1}

Translation



If two successive translation vectors (t_{x1}, t_{y1}) and (t_{x2}, t_{y2}) are applied to a coordinate position **P**, the final transformed location **P'** is calculated as

$$\begin{aligned} \mathbf{P'} &= T(t_{x2}, t_{y2}) \cdot \{\mathbf{T}(t_{x1}, t_{y1}) \cdot \mathbf{P}\} \\ &= \{\mathbf{T}(t_{x2}, t_{y2}) \cdot \mathbf{T}(t_{x1}, t_{y1})\} \cdot \mathbf{P} \end{aligned} \qquad (5\text{-}23)$$

where **P** and **P'** are represented as homogeneous-coordinate column vectors. We can verify this result by calculating the matrix product for the two associative groupings. Also, the composite transformation matrix for this sequence of translations is

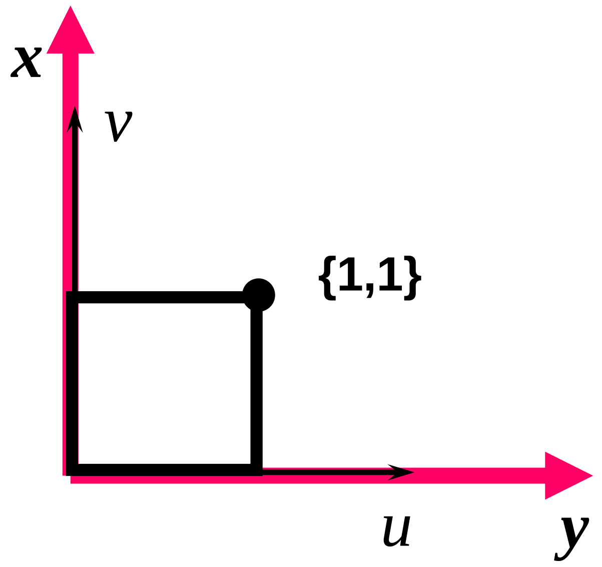
$$\begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}24)$$

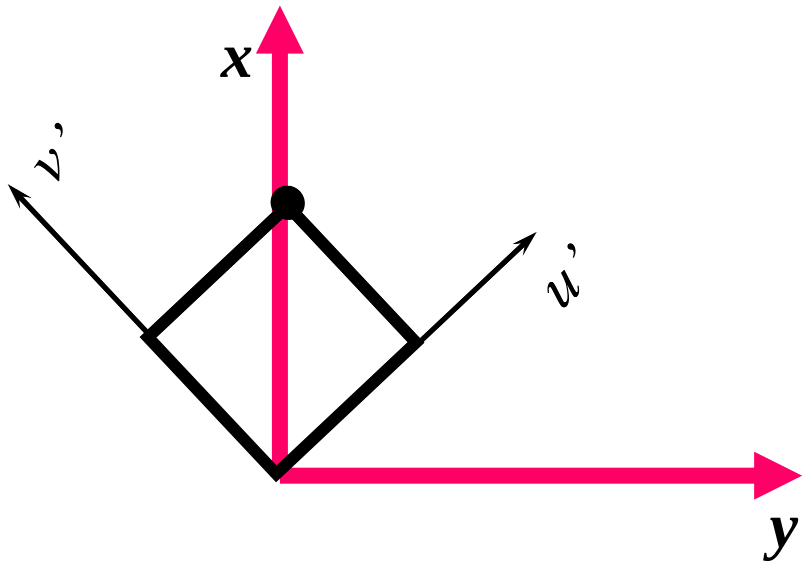
or

$$\mathbf{T}(t_{x2}, t_{y2}) \cdot \mathbf{T}(t_{x1}, t_{y1}) = \mathbf{T}(t_{x1} + t_{x2}, t_{y1} + t_{y2}) \qquad (5\text{-}25)$$

which demonstrates that two successive translations are additive.

Rotation





$$\mathbf{Q} = \begin{bmatrix} \cos\theta & -\sin\theta & \mathbf{0} \\ \sin\theta & \cos\theta & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \mathbf{P}$$

$$v = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 1 \end{bmatrix} \qquad u = \begin{bmatrix} -\sin\theta \\ \cos\theta \\ 1 \end{bmatrix}$$

Rotation

Two successive rotations applied to point **P** produce the transformed position

$$\mathbf{P}' = \mathbf{R}(\theta_2) \cdot \{\mathbf{R}(\theta_1) \cdot \mathbf{P}\}$$
$$= \{\mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1)\} \cdot \mathbf{P} \qquad (5\text{-}26)$$

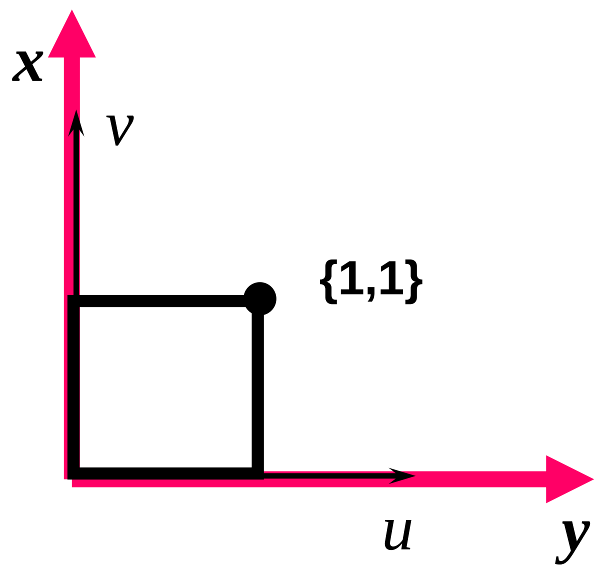
By multiplying the two rotation matrices, we can verify that two successive rotations are additive:

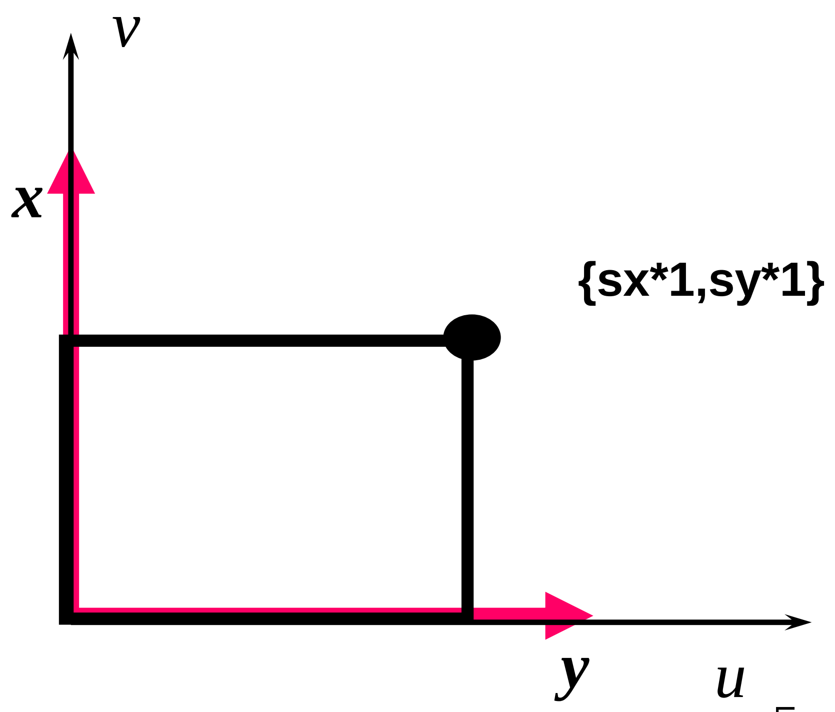
$$\mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1) = \mathbf{R}(\theta_1 + \theta_2) \qquad (5\text{-}27)$$

so that the final rotated coordinates can be calculated with the composite rotation matrix as

$$\mathbf{P}' = \mathbf{R}(\theta_1 + \theta_2) \cdot \mathbf{P} \qquad (5\text{-}28)$$

Scaling





$$\mathbf{Q} = \begin{bmatrix} sx & 0 & \mathbf{0} \\ 0 & sy & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \mathbf{P} \qquad O = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad v = \begin{bmatrix} sx \\ 0 \\ 1 \end{bmatrix} \qquad u = \begin{bmatrix} 0 \\ sy \\ 1 \end{bmatrix}$$

Scaling

Concatenating transformation matrices for two successive scaling operations produces the following composite scaling matrix:

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-29}$$

or

$$\mathbf{S}(s_{x2}, s_{y2}) \cdot \mathbf{S}(s_{x1}, s_{y1}) = \mathbf{S}(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2}) \tag{5-30}$$

The resulting matrix in this case indicates that successive scaling operations are multiplicative. That is, if we were to triple the size of an object twice in succession, the final size would be nine times that of the original.

Shearing

An x-direction shear relative to the x axis is produced with the transformation matrix

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-53}$$

which transforms coordinate positions as

$$x' = x + sh_x \cdot y, \qquad y' = y \tag{5-54}$$

Any real number can be assigned to the shear parameter sh_x. A coordinate position (x, y) is then shifted horizontally by an amount proportional to its distance (y value) from the x axis ($y = 0$). Setting sh_x to 2, for example, changes the square in Fig. 5-23 into a parallelogram. Negative values for sh_x shift coordinate positions to the left.

We can generate x-direction shears relative to other reference lines with

$$\begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{\text{ref}} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-55}$$

with coordinate positions transformed as

$$x' = x + sh_x(y - y_{\text{ref}}), \qquad y' = y \tag{5-56}$$

Shearing

A y-direction shear relative to the line $x = x_{ref}$ is generated with the transformation matrix

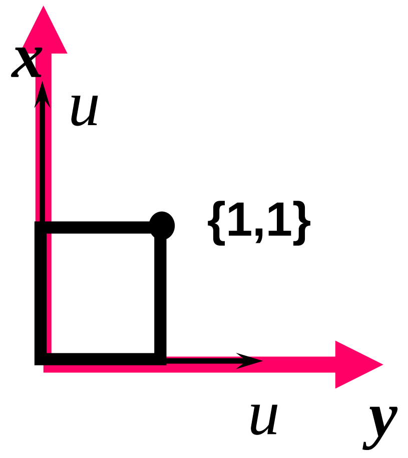
$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix} \tag{5-57}$$

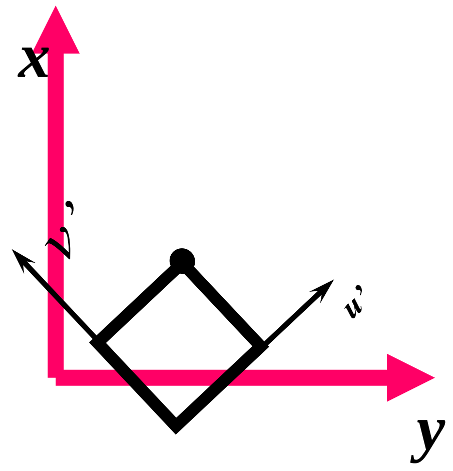
which generates transformed coordinate positions

$$x' = x, \qquad y' = sh_y(x - x_{ref}) + y \tag{5-58}$$

This transformation shifts a coordinate position vertically by an amount proportional to its distance from the reference line $x = x_{ref}$. Figure 5-25 illustrates the conversion of a square into a parallelogram with $sh_y = 1/2$ and $x_{ref} = -1$.

Composite Transformations





$$O = \begin{bmatrix} x_1(1-\cos\theta) + y_1\sin\theta \\ y_1(1-\cos\theta) - y_1\sin\theta \\ 1 \end{bmatrix}$$

$$v = \begin{bmatrix} \cos\theta + x_1(1-\cos\theta) + y_1\sin\theta \\ \sin\theta + y_1(1-\cos\theta) - y_1\sin\theta \\ 1 \end{bmatrix}$$

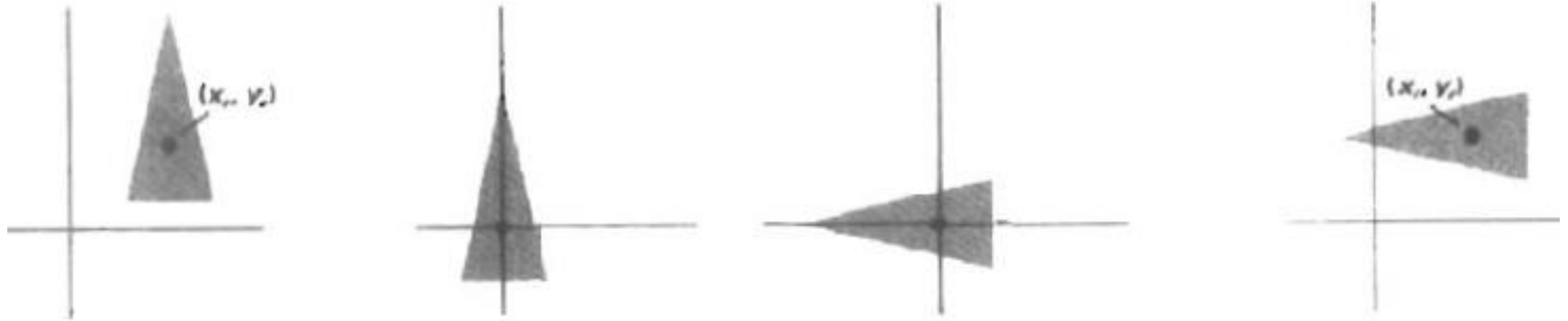
$$M = \begin{bmatrix} \cos\theta & -\sin\theta & x_1(1-\cos\theta) + y_1\sin\theta \\ \sin\theta & \cos\theta & y_1(1-\cos\theta) - x_1\sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

$$u = \begin{bmatrix} -\sin\theta + x_1(1-\cos\theta) + y_1\sin\theta \\ \cos\theta + y_1(1-\cos\theta) - y_1\sin\theta \\ 1 \end{bmatrix}$$

General Pivot-Point Rotation

With a graphics package that only provides a rotate function for revolving objects about the coordinate origin, we can generate rotations about any selected pivot point (x_r, y_r) by performing the following sequence of translate–rotate–translate operations:

1. Translate the object so that the pivot-point position is moved to the coordinate origin.
2. Rotate the object about the coordinate origin.
3. Translate the object so that the pivot point is returned to its original position.



(a) Original Position of Object and Pivot Point

(b) Translation of Object so that Pivot Point (x_r, y_r) is at Origin

(c) Rotation about Origin

(d) Translation of Object so that the Pivot Point is Returned to Position (x_r, y_r)

General Pivot-Point Rotation

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

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r\sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r\sin\theta \\ 0 & 0 & 1 \end{bmatrix} \tag{5-31}$$

which can be expressed in the form

$$\mathbf{T}(x_r, y_r) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(-x_r, -y_r) = \mathbf{R}(x_r, y_r, \theta) \tag{5-32}$$

where $\mathbf{T}(-x_r, -y_r) = \mathbf{T}^{-1}(x_r, y_r)$. In general, a rotate function can be set up to accept parameters for pivot-point coordinates, as well as the rotation angle, and to generate automatically the rotation matrix of Eq. 5-31.

General Fixed-Point Rotation

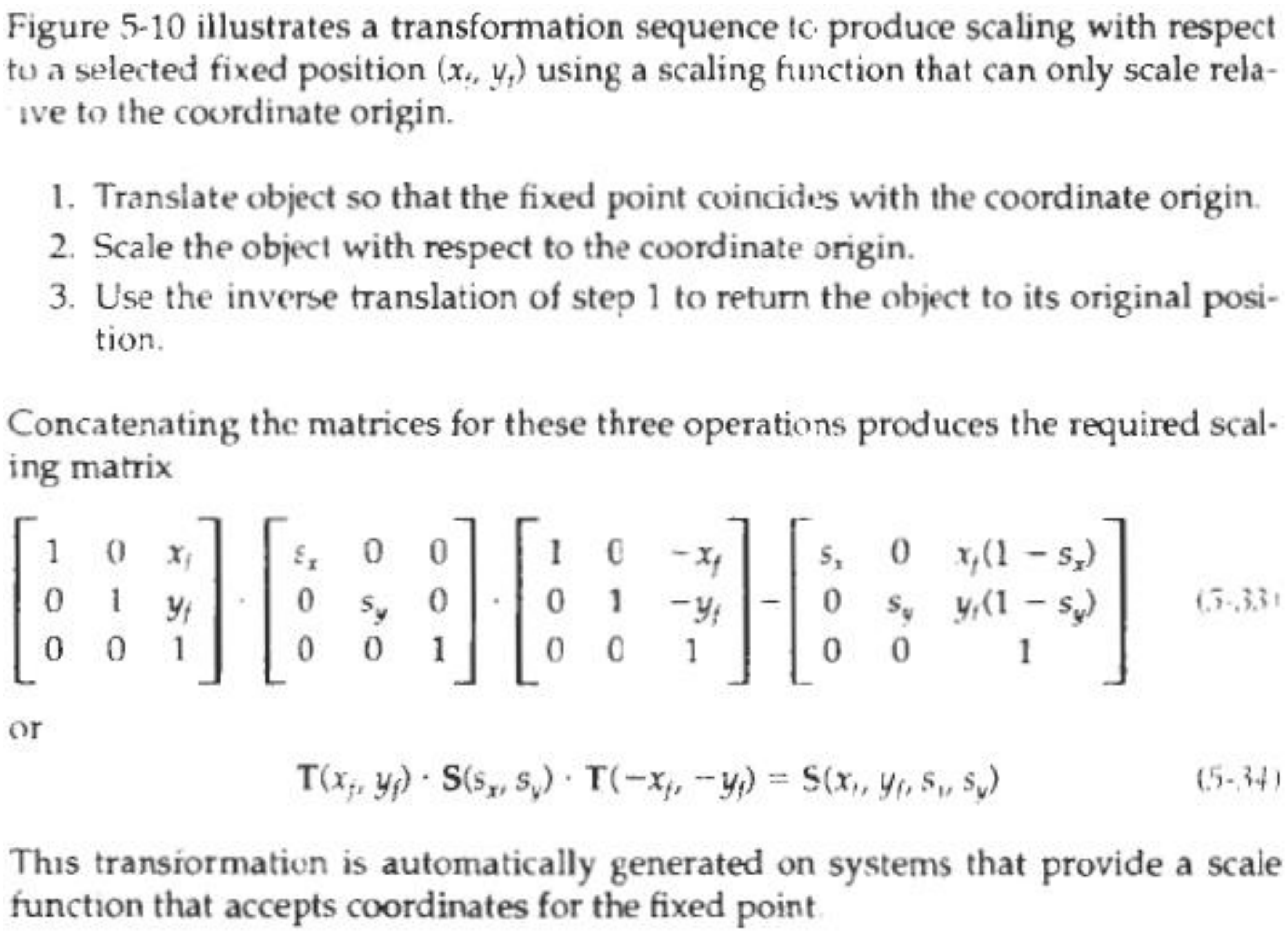


Figure 5-10 illustrates a transformation sequence to produce scaling with respect to a selected fixed position (x_f, y_f) using a scaling function that can only scale relative to the coordinate origin.

1. Translate object so that the fixed point coincides with the coordinate origin.
2. Scale the object with respect to the coordinate origin.
3. Use the inverse translation of step 1 to return the object to its original position.

Concatenating the matrices for these three operations produces the required scaling matrix

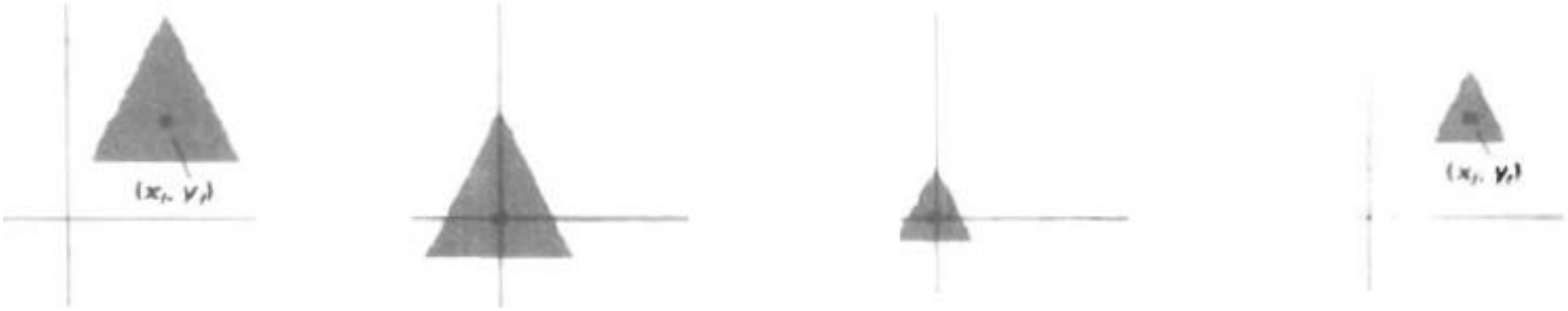
$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_f(1 - s_x) \\ 0 & s_y & y_f(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}33)$$

or

$$\mathbf{T}(x_f, y_f) \cdot \mathbf{S}(s_x, s_y) \cdot \mathbf{T}(-x_f, -y_f) = \mathbf{S}(x_f, y_f, s_x, s_y) \qquad (5\text{-}34)$$

This transformation is automatically generated on systems that provide a scale function that accepts coordinates for the fixed point.

General Fixed-Point Rotation



(a)
Original Position of Object and Fixed Point

(b)
Translate Object so that Fixed Point (x_f, y_f) is at Origin

(c)
Scale Object with Respect to Origin

(d)
Translate Object so that the Fixed Point is Returned to Position (x_f, y_f)

General Composite Transformation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} rs_{xx} & rs_{xy} & trs_x \\ rs_{yx} & rs_{yy} & trs_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{T}(t_x, t_y) \cdot \mathbf{R}(x_c, y_c, \theta) \cdot \mathbf{S}(x_c, y_c, s_x, s_y)$$

$$= \begin{bmatrix} s_x \cos\theta & -s_y \sin\theta & x_c(1 - s_x \cos\theta) + y_c s_y \sin\theta + t_x \\ s_x \sin\theta & s_y \cos\theta & y_c(1 - s_y \cos\theta) - x_c s_x \sin\theta + t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$x' = x \cdot rs_{xx} + y \cdot rs_{xy} + trs_x, \qquad y' = x \cdot rs_{yx} + y \cdot rs_{yy} + trs_y$$

General Composite Transformation

A general **rigid-body transformation matrix**, involving only translations and rotations, can be expressed in the form

$$\begin{bmatrix} r_{xx} & r_{xy} & tr_x \\ r_{yx} & r_{yy} & tr_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5-40}$$

where the four elements r_{ij} are the multiplicative rotation terms, and elements tr_x and tr_y are the translational terms. A rigid-body change in coordinate position is also sometimes referred to as a **rigid-motion** transformation. All angles and distances between coordinate positions are unchanged by the transformation. In addition, matrix 5-40 has the property that its upper-left 2-by-2 submatrix is an orthogonal matrix. This means that if we consider each row of the submatrix as a vector, then the two vectors (r_{xx}, r_{xy}) and (r_{yx}, r_{yy}) form an orthogonal set of unit vectors: Each vector has unit length

$$r_{xx}^2 + r_{xy}^2 = r_{yx}^2 + r_{yy}^2 = 1 \tag{5-41}$$

and the vectors are perpendicular (their dot product is 0):

$$r_{xx}r_{yx} + r_{xy}r_{yy} = 0 \tag{5-42}$$

General Composite Transformation

Therefore, if these unit vectors are transformed by the rotation submatrix, (r_{xx}, r_{xy}) is converted to a unit vector along the x axis and (r_{yx}, r_{yy}) is transformed into a unit vector along the y axis of the coordinate system:

$$\begin{bmatrix} r_{xx} & r_{xy} & 0 \\ r_{yx} & r_{yy} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{xx} \\ r_{xy} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \tag{5-43}$$

$$\begin{bmatrix} r_{xx} & r_{xy} & 0 \\ r_{yx} & r_{yy} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{yx} \\ r_{yy} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \tag{5-44}$$

As an example, the following rigid-body transformation first rotates an object through an angle θ about a pivot point (x_r, y_r) and then translates:

$$\mathbf{T}(t_x, t_y) \cdot \mathbf{R}(x_r, y_r, \theta) = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r\sin\theta + t_x \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r\sin\theta + t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5-45}$$

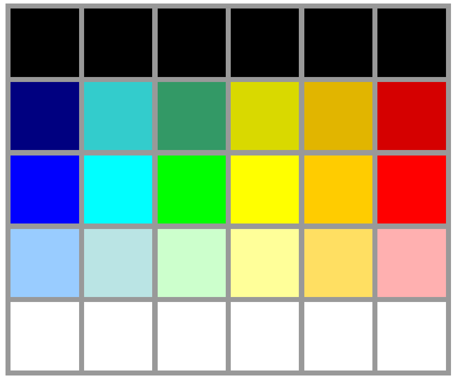
General Composite Transformation

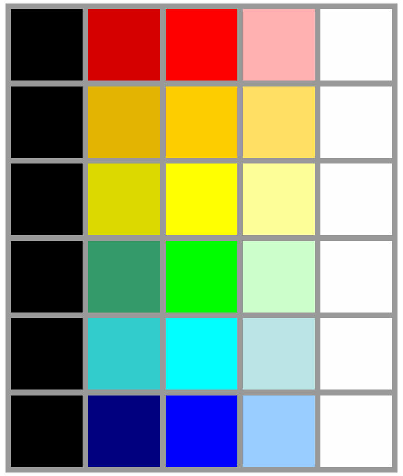
Here, orthogonal unit vectors in the upper-left 2-by-2 submatrix are $(\cos\theta, -\sin\theta)$ and $(\sin\theta, \cos\theta)$, and

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta \\ -\sin\theta \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \tag{5-46}$$

Similarly, unit vector $(\sin\theta, \cos\theta)$ is converted by the transformation matrix in Eq. 5-46 to the unit vector $(0, 1)$ in the y direction.

Transforming Images



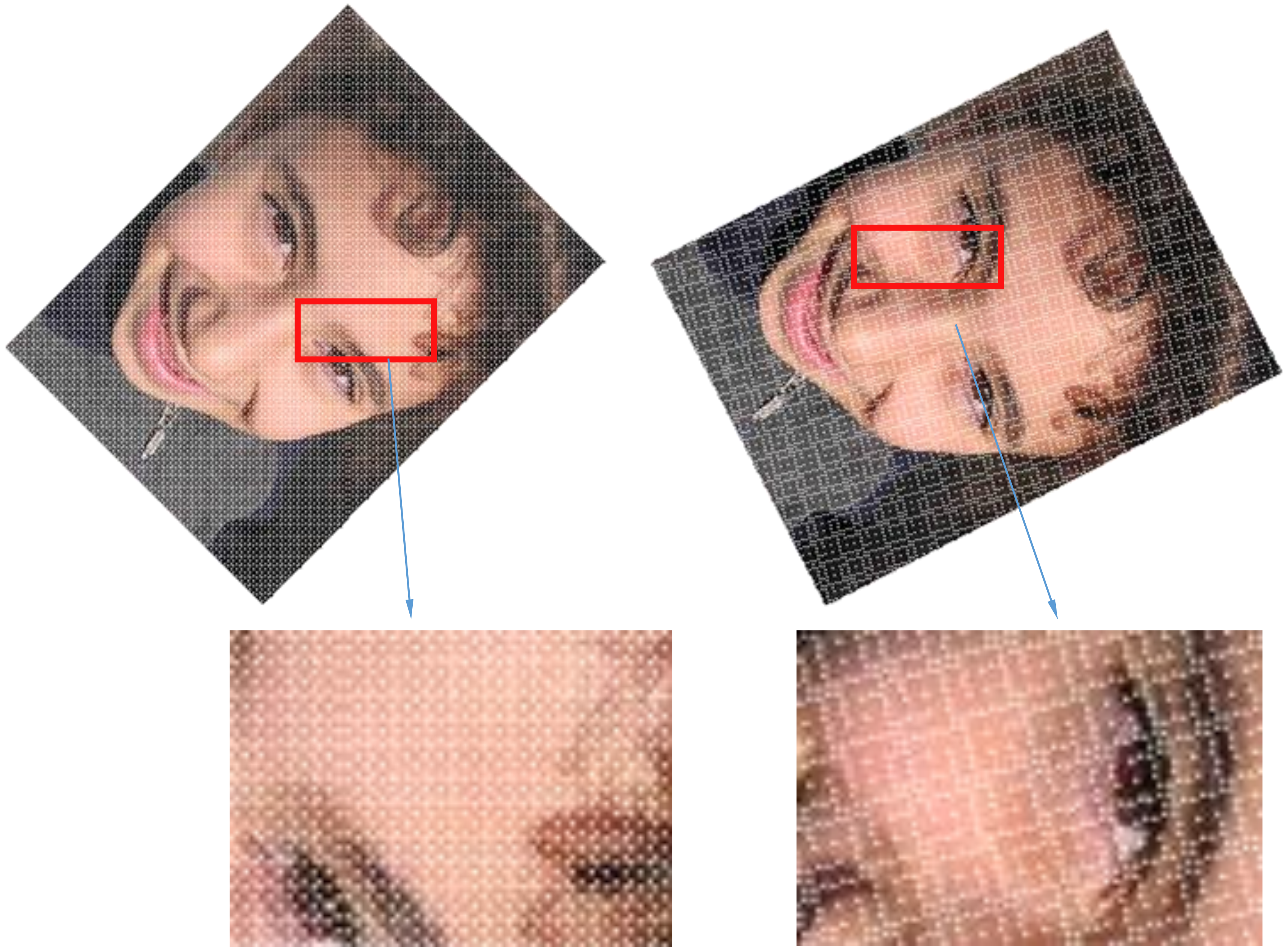


- Simply juggling array positions can achieve simple 90 degree rotations

```
for (int x=0; x<width; x++)
for (int y=0; y<height; y++)
{
        set(x, y,    get (y, x)   );
}
```

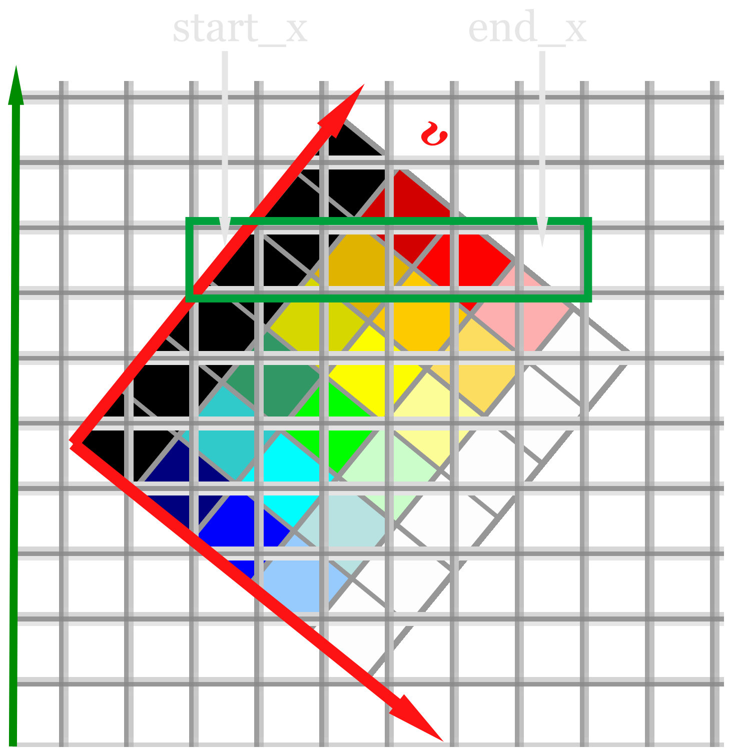
Transformations on Pixels

- Alternatively we could apply a rotation transfrom for each pixel.
- This works in some cases but there are problems when the angle of rotation is not a multiple of 90 degrees
- There often isn’t a 1-to-1 correspondence between number of original and transformed pixels.
- In Addition, rounding to the nearest discrete block can create errors that lead to gaps in the transformed image.



Transforming Images

- The solution is to look at it from the point of view of rasterizing a transformed polygon which is textured with the original image



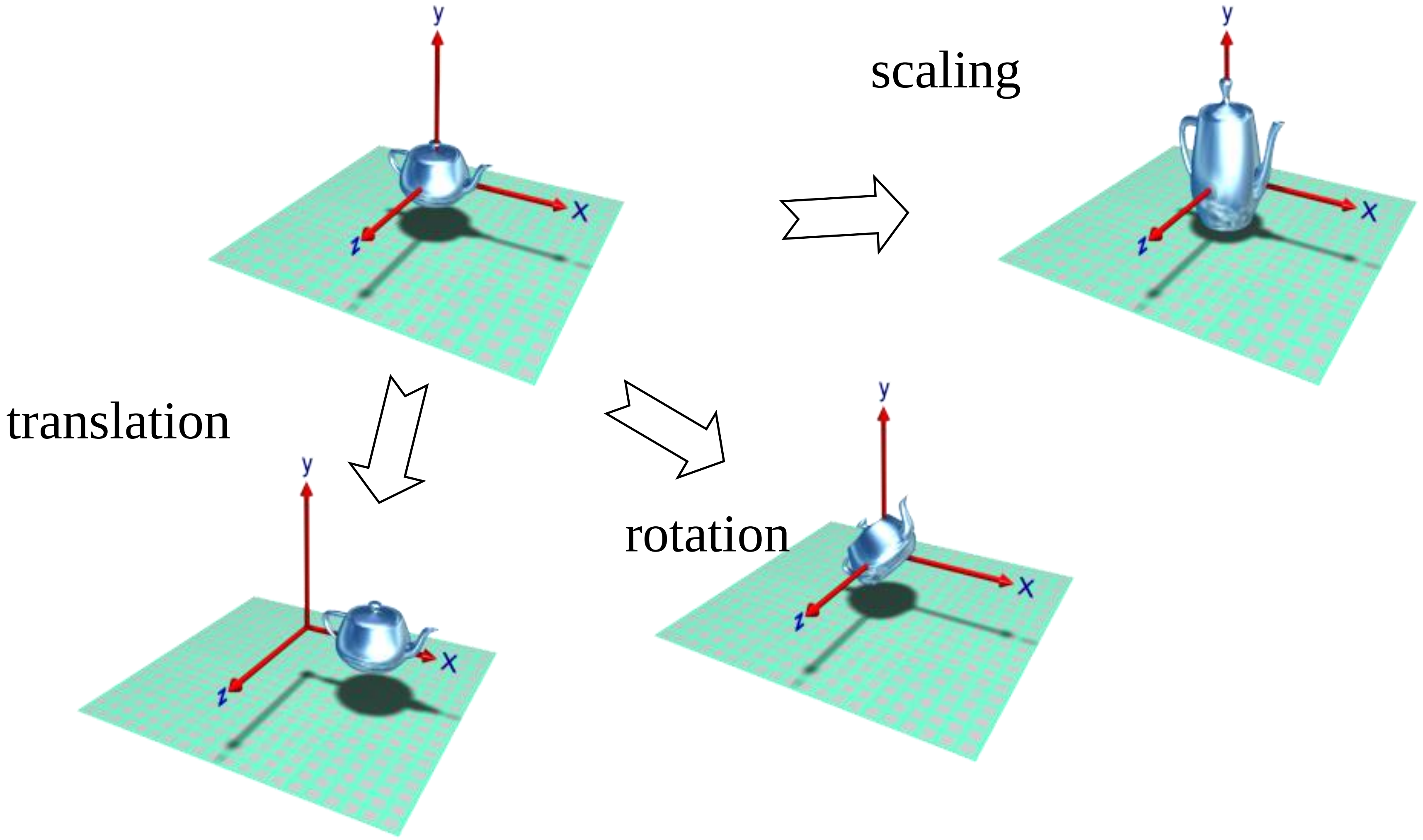
- **ROTATE IMAGE by θ :**

- Find rotated "frame" of the image
- For each scanline (y)
  - Using interpolation find **startx** and **endx**
  - For every pixel x (between **startx** and **endx**)
    - Find {u,v} corresponding to {x, y}
    - This is done by rotating {x,y} by -θ
    - set ({x,y}, get {u,v})

- This ensures that every pixel in the rotated image is dealt with and no gaps are left

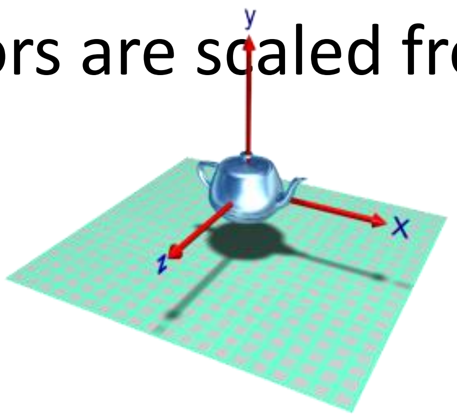
N.B. In most cases there will be some distortion in the image. So rotating images by arbitrary angles can cause degradation.

3D Object Transformations

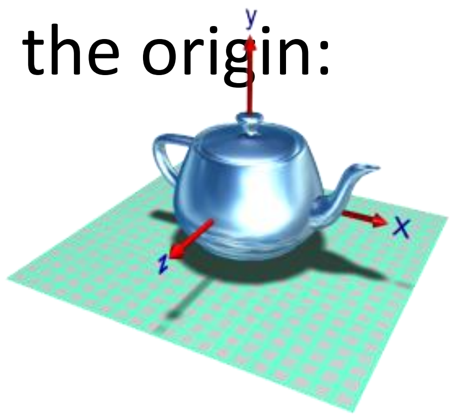


Scale

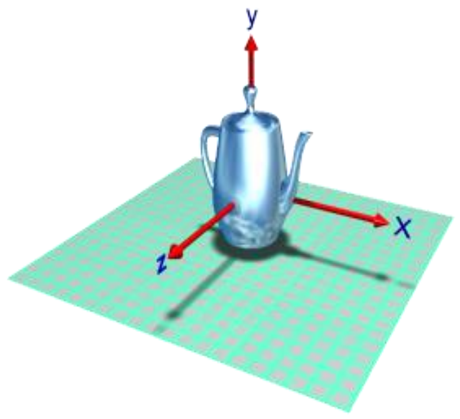
- all vectors are scaled from the origin:



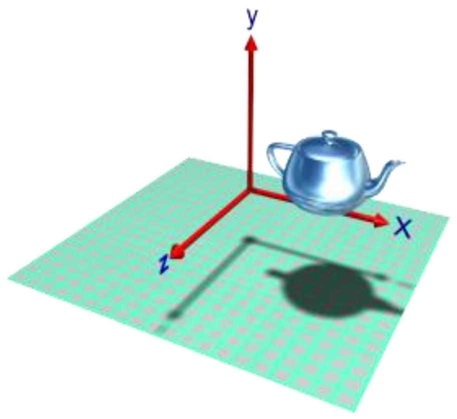
Original



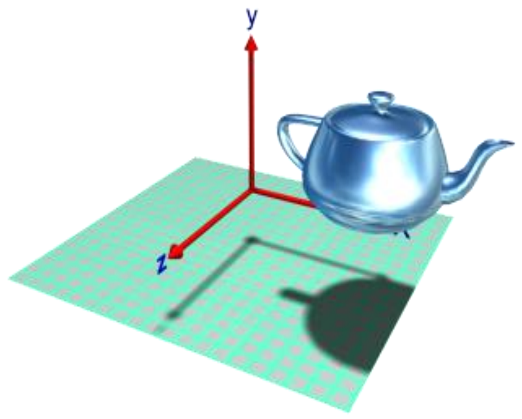
scale all axes



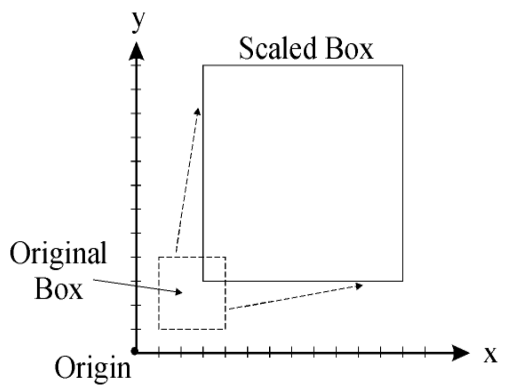
scale Y axis



offset from origin



distance from origin also scales



Scale

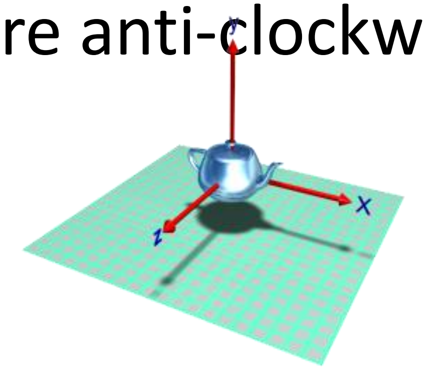
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \Rightarrow \quad \mathbf{q} = \mathbf{Sp}$$

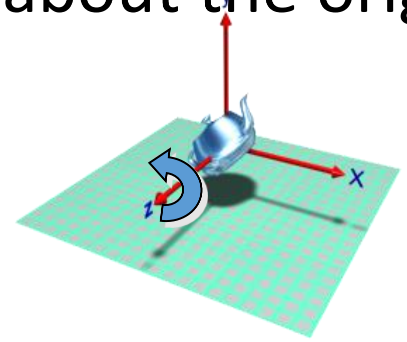
Or in 3D *homogeneous coordinates*

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \\ w \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

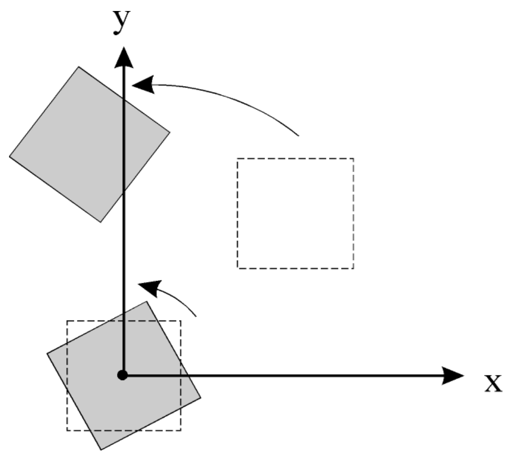
Rotation

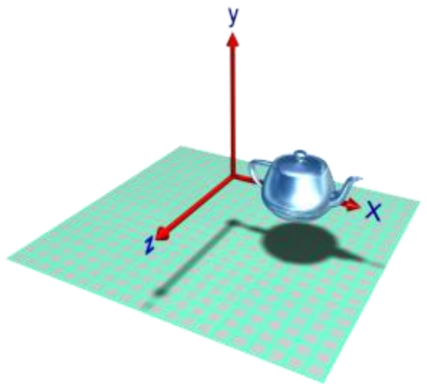
- Rotations are anti-clockwise about the origin:

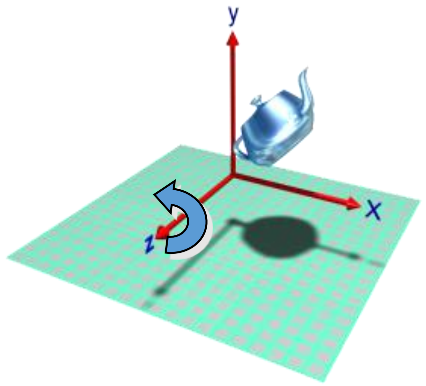




rotation of 45^{o} about the Z axis







offset from origin rotation

Rotation

- 2D rotation of θ about origin:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

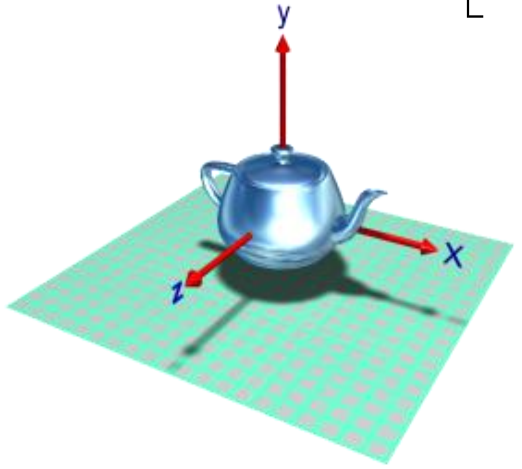
- 3D homogeneous rotations:

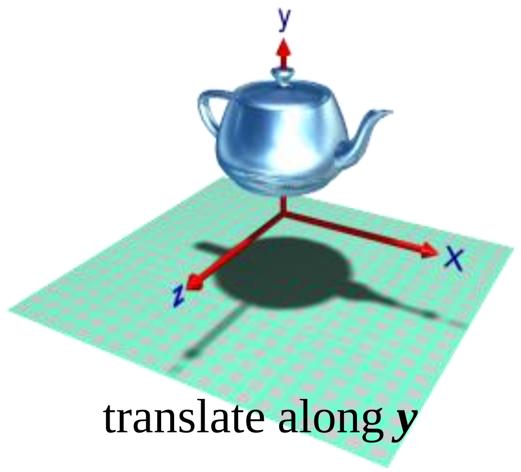
$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation

- Translation only applies to points, we never translate vectors.
- Remember: points have homogeneous co-ordinate w = 1

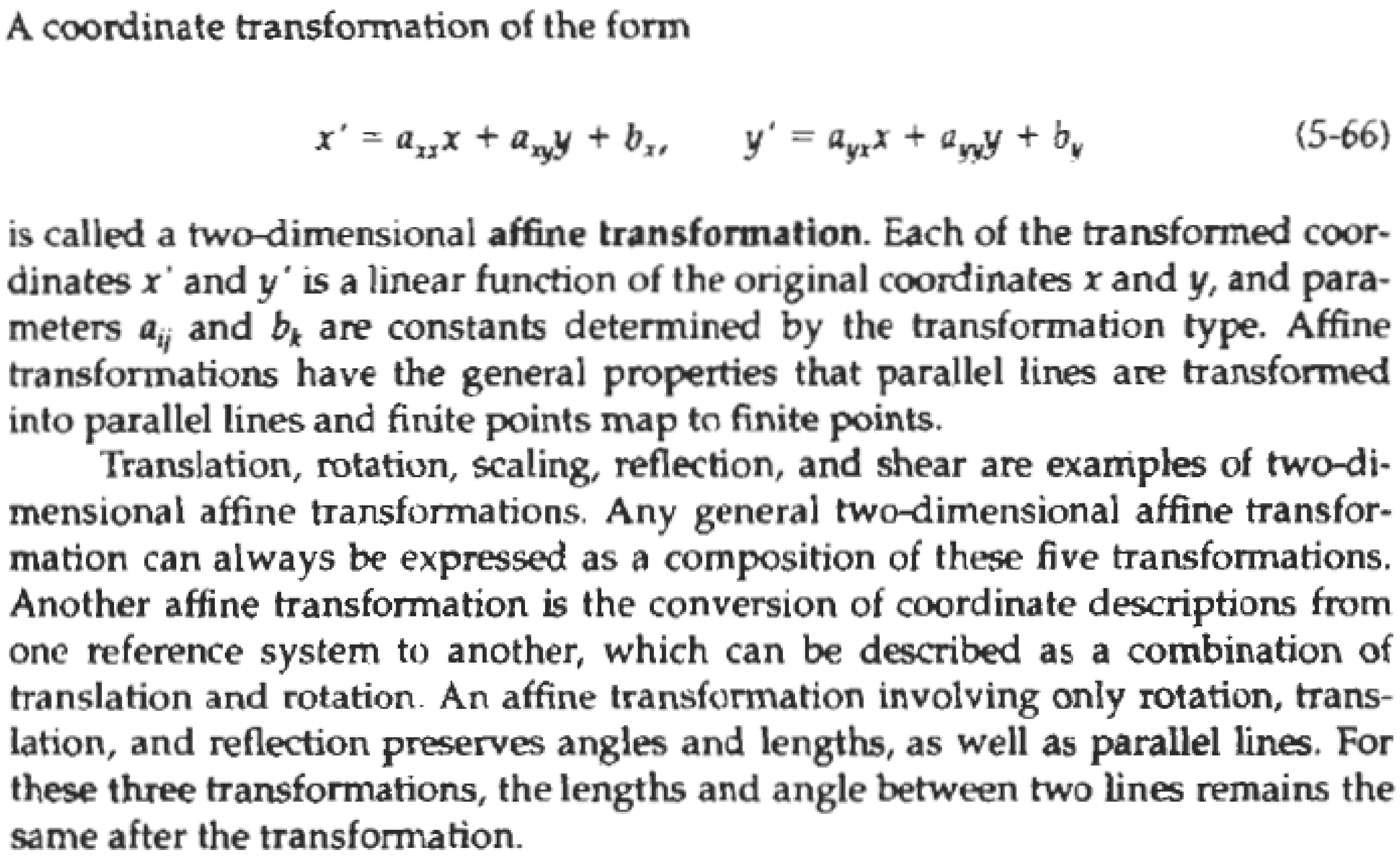
$$
\begin{aligned}
x' &= x + a \\
y' &= y + b \\
z' &= z + c
\end{aligned}
\quad \Rightarrow \quad
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \\ z+c \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
$$





translate along *y*

Affine Transformation



A coordinate transformation of the form

$$x' = a_{xx}x + a_{xy}y + b_x, \qquad y' = a_{yx}x + a_{yy}y + b_y \tag{5-66}$$

is called a two-dimensional **affine transformation**. Each of the transformed coordinates x' and y' is a linear function of the original coordinates x and y, and parameters a_{ij} and b_k are constants determined by the transformation type. Affine transformations have the general properties that parallel lines are transformed into parallel lines and finite points map to finite points.

Translation, rotation, scaling, reflection, and shear are examples of two-dimensional affine transformations. Any general two-dimensional affine transformation can always be expressed as a composition of these five transformations. Another affine transformation is the conversion of coordinate descriptions from one reference system to another, which can be described as a combination of translation and rotation. An affine transformation involving only rotation, translation, and reflection preserves angles and lengths, as well as parallel lines. For these three transformations, the lengths and angle between two lines remains the same after the transformation.