



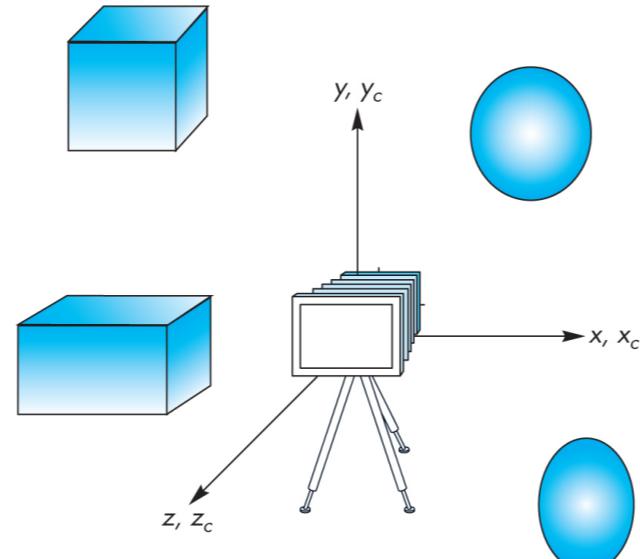
# Geometric Transformations

---

Introduction to Computer Graphics  
CSE 533/333

# Transformation Matrices

- Fundamental to computer graphics
  - Object modelling
  - Viewing
  - Projecting



Source: <http://pcwieczkowski.pl>

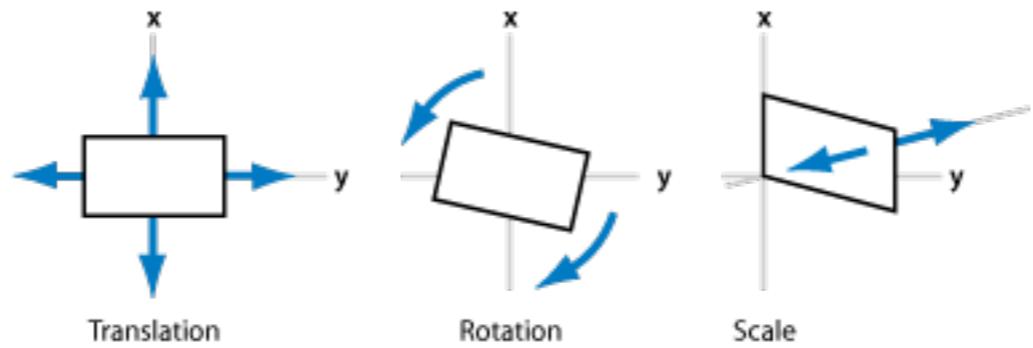


Source: <http://en.wikipedia.org/wiki/Anamorphosis>

# Using Transformations

## *Object*

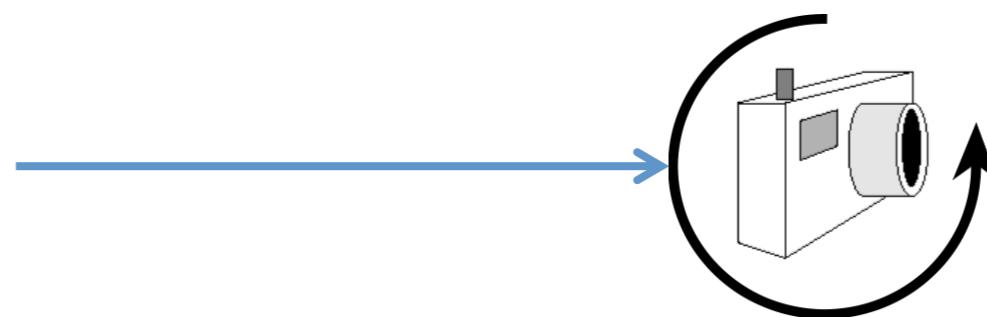
- Objects in a scene are collection of points
- Objects have location, orientation and size
- Correspond to transformations: Translation (**T**), Rotation (**R**), and Scaling (**S**)



# Using Transformations

## *Camera*

- Scene has a camera/view-point/eye that views the scene
- The camera has a location and orientation
- Correspond to transformations: Translation (**T**), Rotation (**R**)



# Linear Transformations

- Can be represented as invertible (non-singular) matrices
- In 2D, can be represented by  $2 \times 2$  matrices:

$$T = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

- The basic transform looks like:

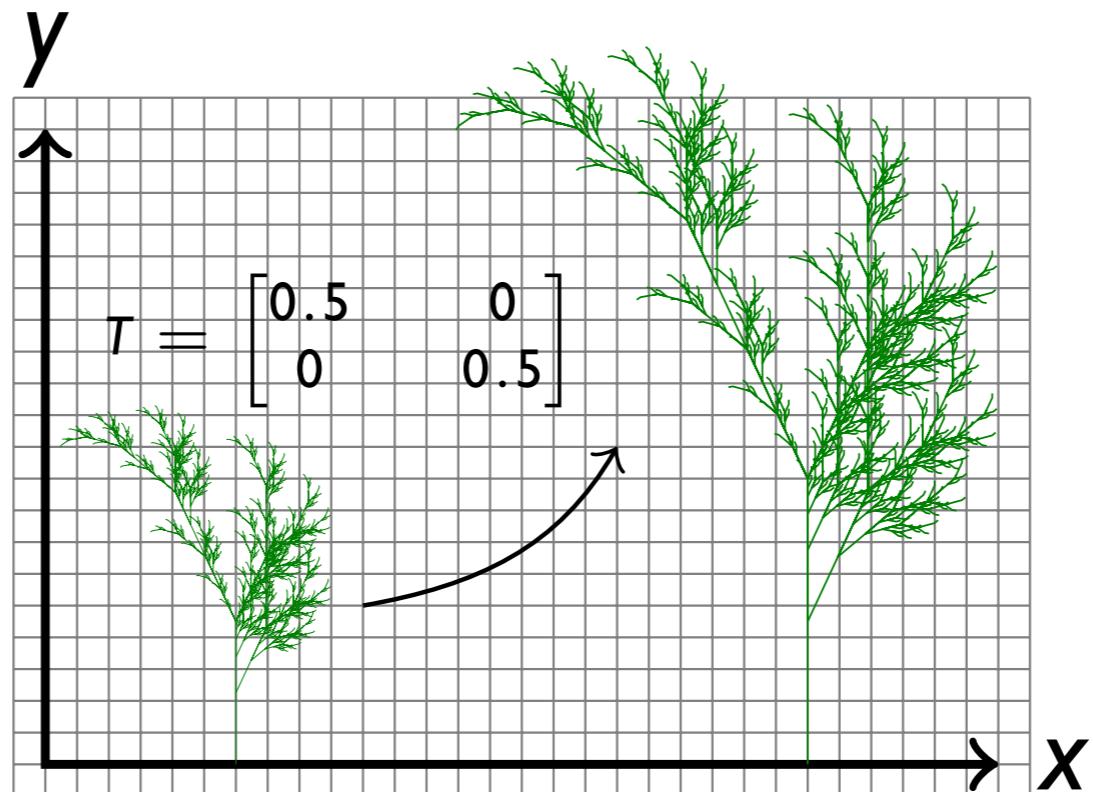
$$Tp = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

# 2D Transformations

# Scaling (2D)

- Non-uniform scaling

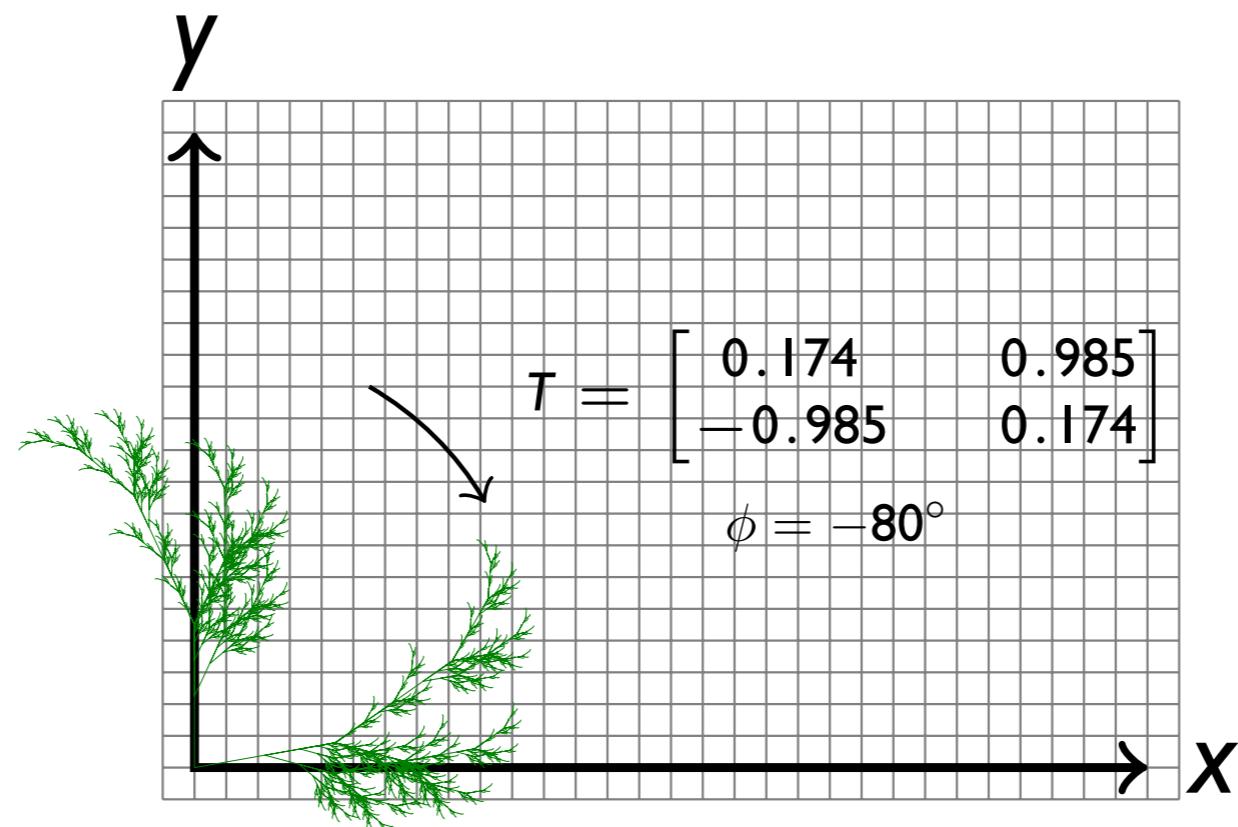
$$\text{scale}(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$



# Rotation (2D)

- Rotation about origin

$$\text{rotate}(\phi) = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}$$



# Translation (2D)

- Possible to write in a  $2 \times 2$  matrix?
  - No!
- Translation is not a linear transform  
(use linearity properties to show)

# Non-linearity of Translation

- Linearity conditions:

$$L(p + q) = L(p) + L(q)$$

$$L(ap) = aL(p)$$

- For translation:

$$T(p) = p + t$$

$$T(p + q) = p + q + t \neq (p + t) + (q + t) = T(p) + T(q),$$

$$T(ap) = ap + t \neq a(p + t) = aT(p)$$

# Translation (2D)

- Use **Homogeneous coordinates**
  - Add an additional dimension, the w-axis, and an extra coordinate, the w-component.
  - Effectively the hyperspace for embedding 2D space
- Clever trick to incorporate translations into a single matrix of transformations

$$\begin{bmatrix} m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1 \end{bmatrix}$$

# Translation (2D)

- The transformation now looks like

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}x + m_{12}y + x_t \\ m_{21}x + m_{22}y + y_t \\ 1 \end{bmatrix}$$

- For positions this is fine, but vectors (or displacements) cannot be translated.
  - Make the third coordinate zero!

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

is a location, while

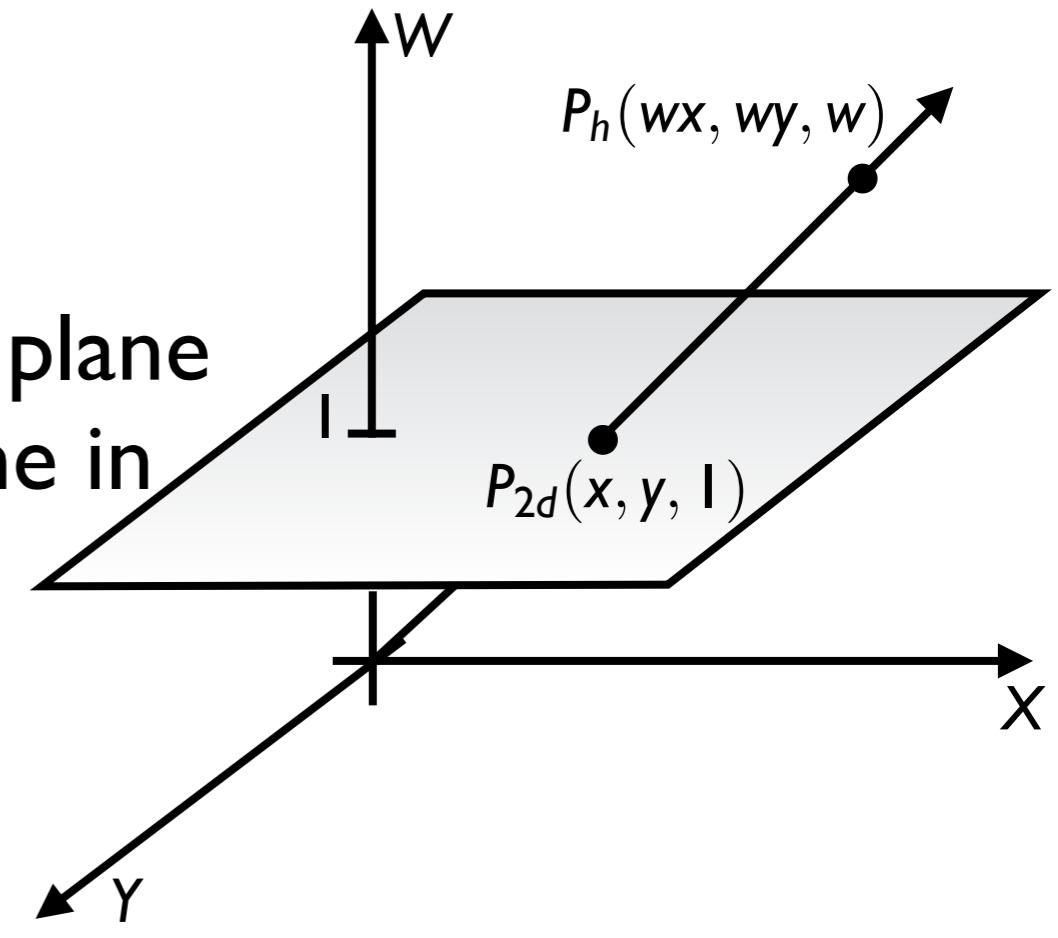
$$\begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

is a vector/direction

# Homogeneous Coordinates

- Allow expressing all three 2D transformations as  $3 \times 3$  matrices
- We start with point  $P_{2d}$  on the XY plane and map it to bring it to the w-plane in the hyperspace

$$P_{2d}(x, y) \mapsto P_h(wx, wy, w), w \neq 0$$

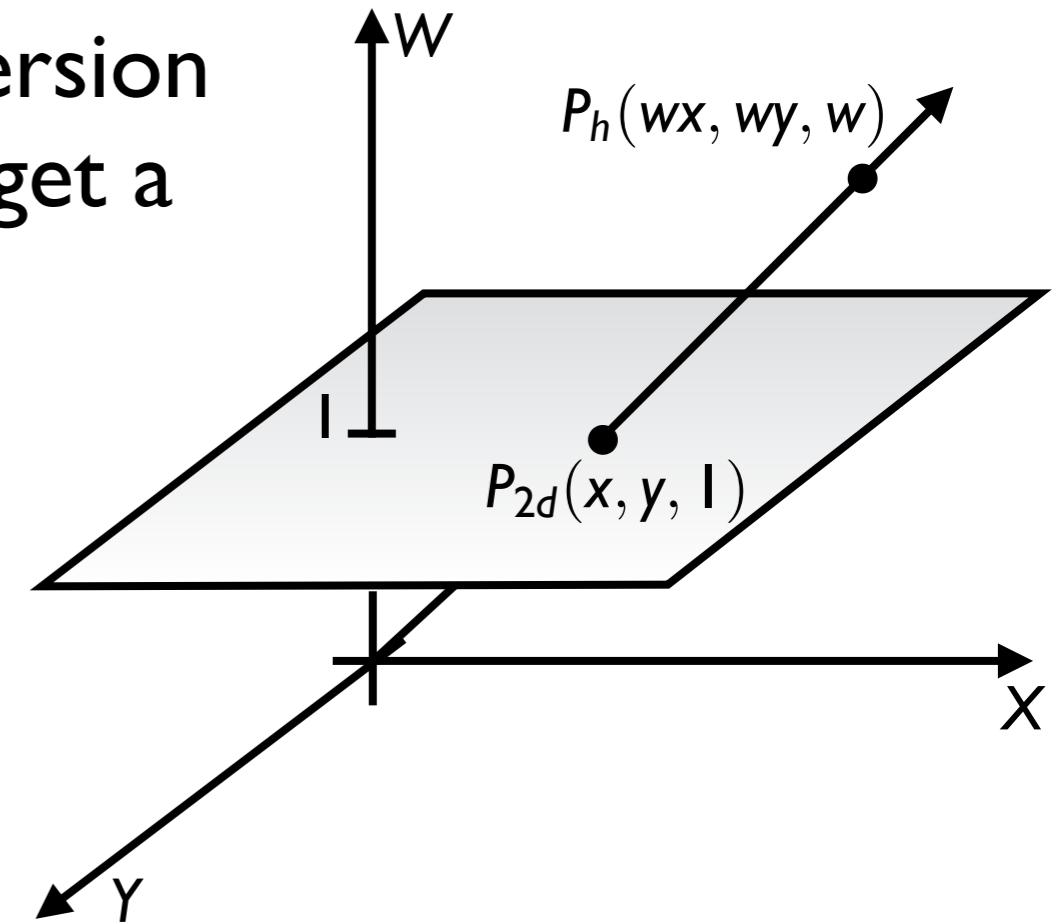


- The resulting  $(x', y')$  coordinates in our new point  $P_h$  are different from the original  $(x, y)$ , since  $x' = wx, y' = wy$

# Homogeneous Coordinates

- We can now apply homogenised version of our transformation matrices to get a new point in the hyperspace
- To get the corresponding 2D point back, we divide all the three coordinates by the  $w$ -component
- In effect, we want our  $3 \times 3$  transformation matrix to map points

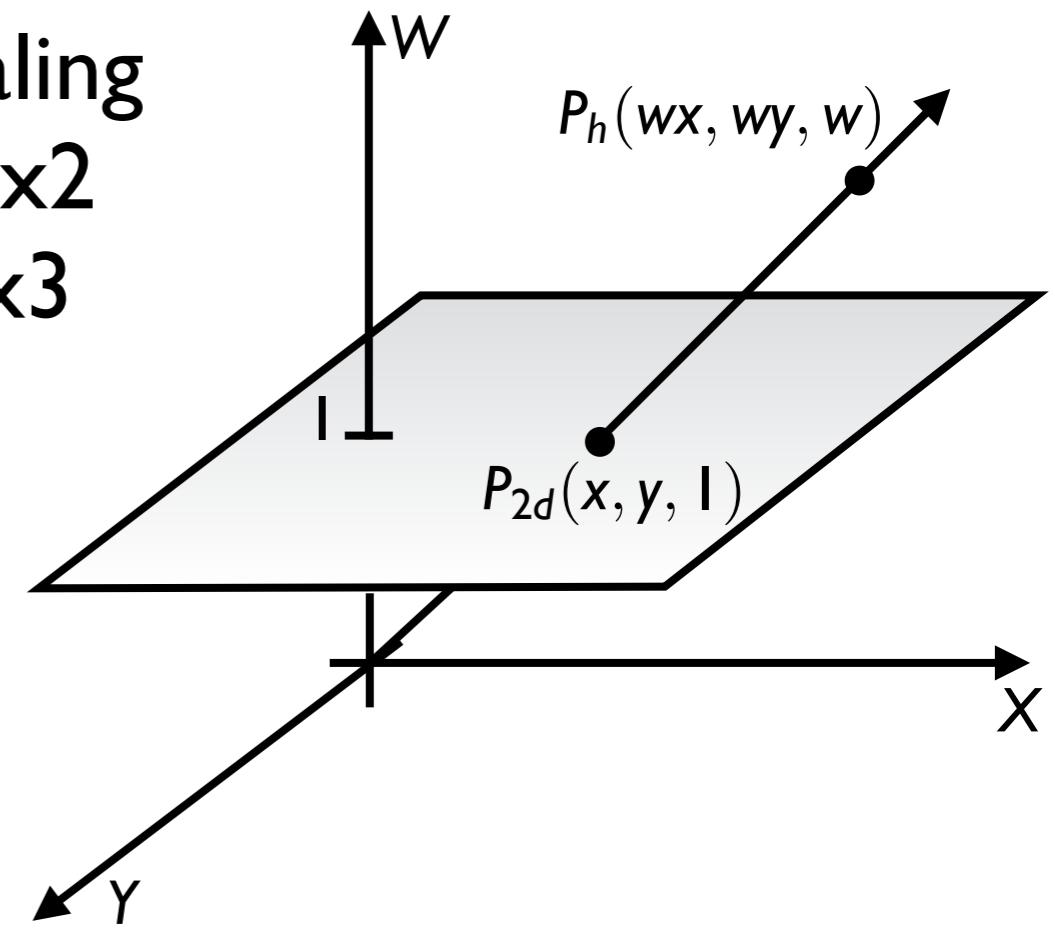
$$v = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ to points } v' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$



# Homogeneous Coordinates

- For linear transformations ( i.e., scaling and rotation), embed the existing  $2 \times 2$  matrix in the upper-left of a new  $3 \times 3$  matrix:

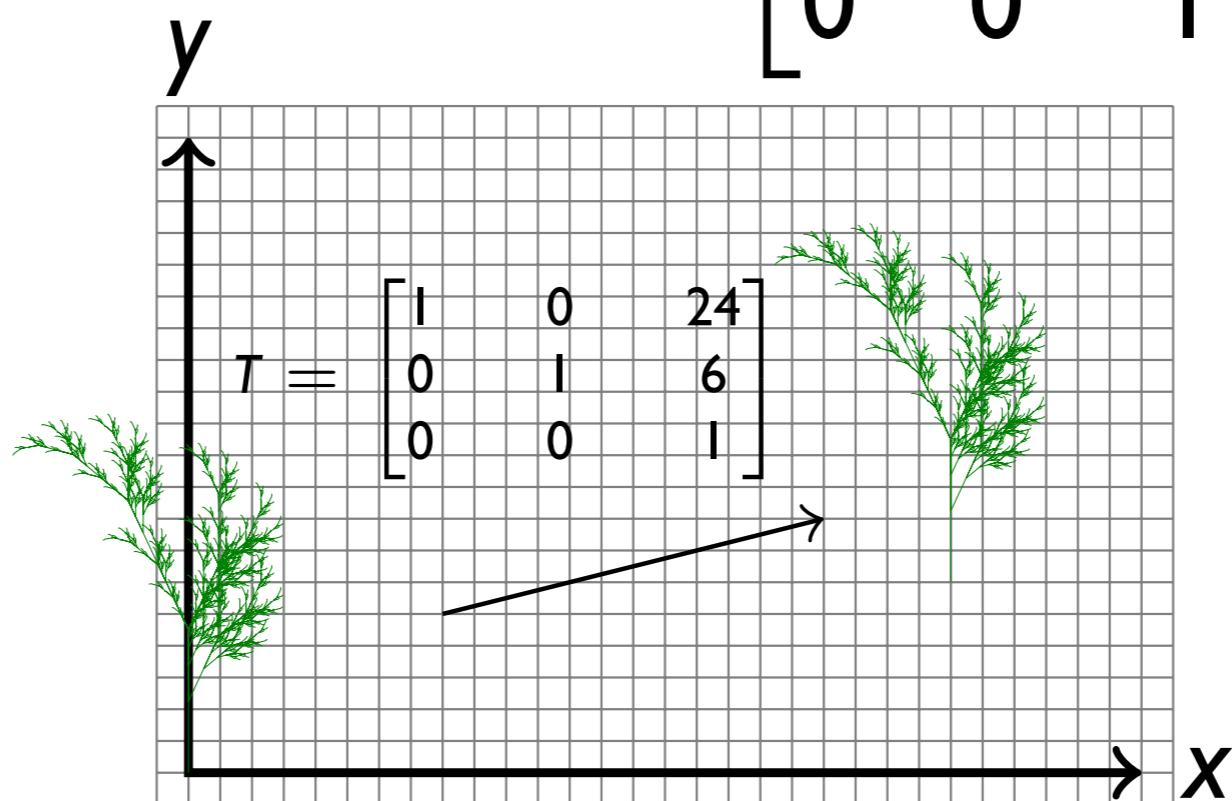
$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



# Translation (2D)

- Translation matrix can now be represented by embedding the translation vector in the right column

$$\text{translate}(dx, dy) = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$

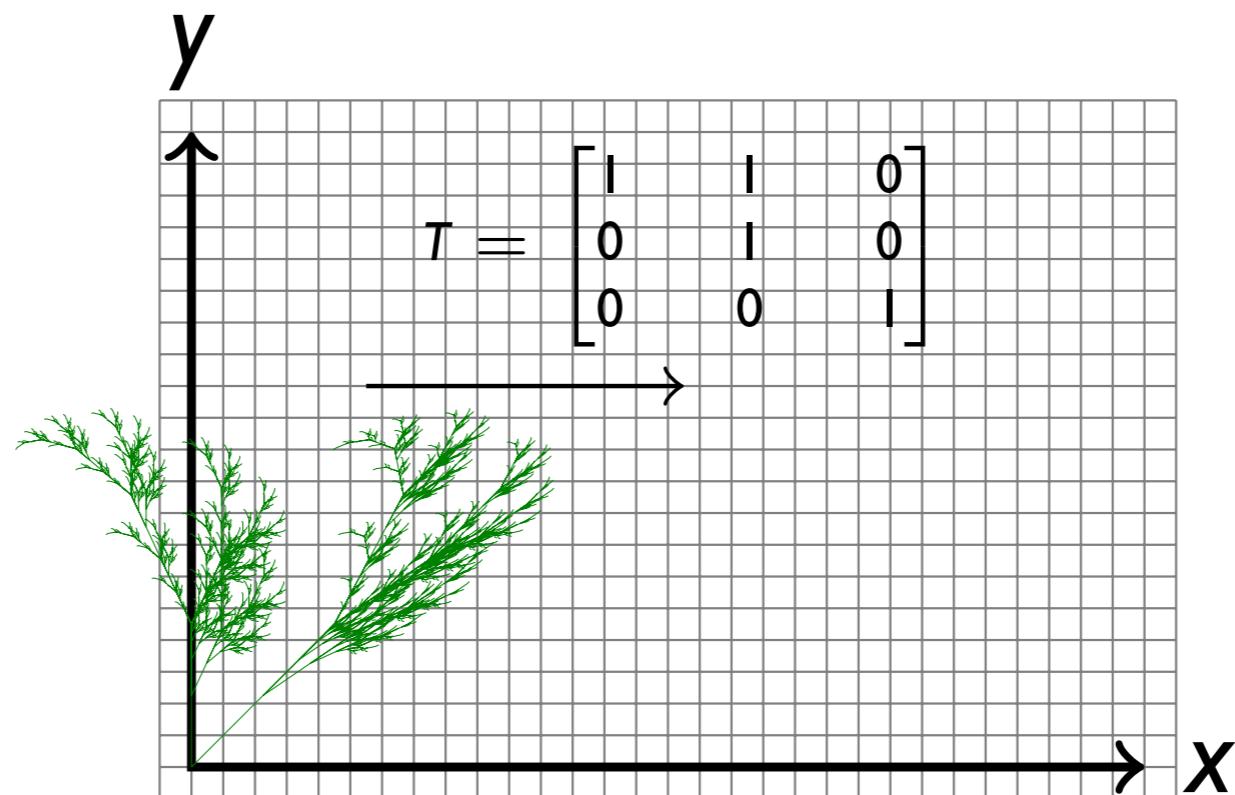


# Shearing (2D)

- Skew an object to the side

$$\text{shear}_x = \begin{bmatrix} 1 & \tan \phi & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{shear}_y = \begin{bmatrix} 1 & 0 & 0 \\ \tan \phi & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\phi$  is angle  
with the axis



# Transformations

## Homogenised

Transformation	Matrix
Scaling	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Rotation	$\begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Translation	$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$

Note: These transformations are called **affine** transformations.

# Inverse Transformations

Transformation	Matrix Inverse
Scaling	$\begin{bmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Rotation	$\begin{bmatrix} \cos \phi & \sin \phi & 0 \\ -\sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Translation	$\begin{bmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{bmatrix}$

# Composition of Transformations

- Transformation is a function; by associativity, we can compose functions
$$(f \circ g)(x) \equiv f(g(x))$$
- Given transformations  $M_1, M_2$  and input vertex  $v$ , our composition is equivalent to
$$v' = M_1 M_2 v$$

# Composition of Transformations

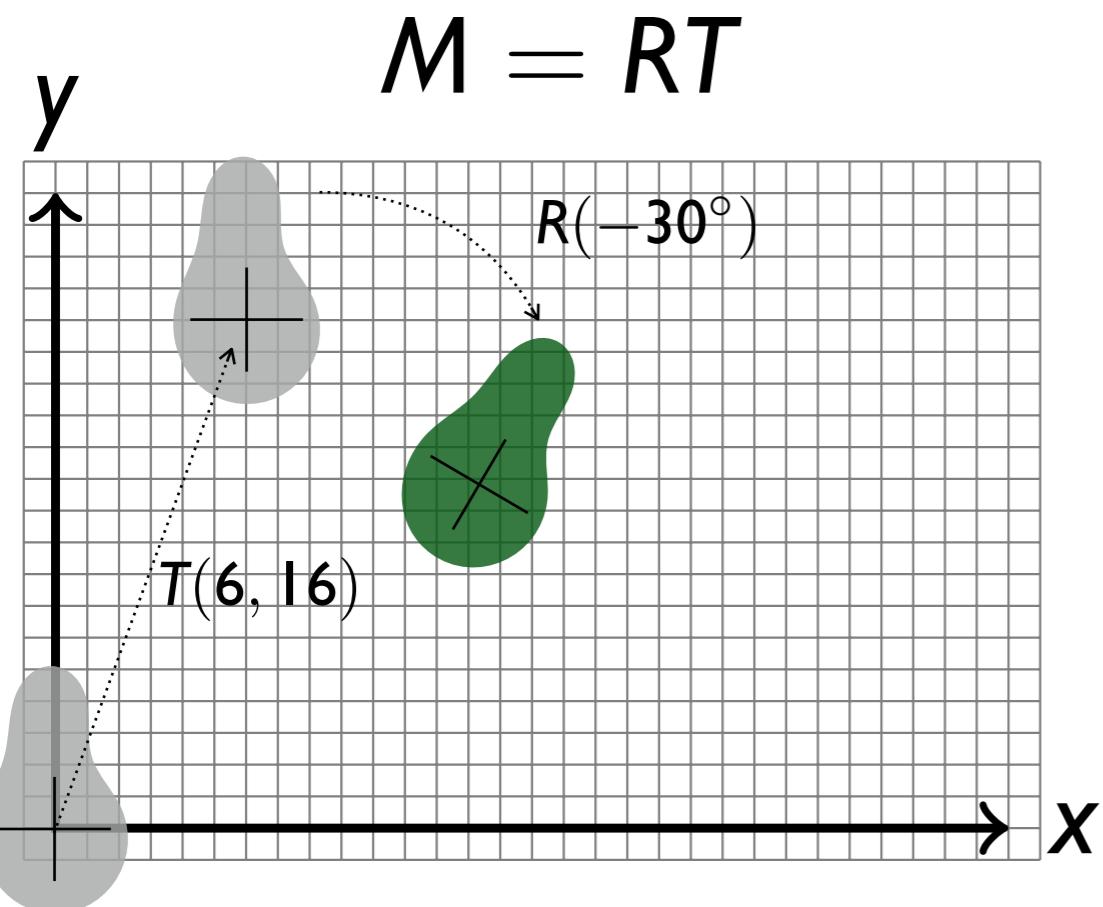
- Transformations can be combined to form a more complex transformation
- E.g.:  $M = TRS$ , which scales a point, then rotates it, and then translates it:

$$\underbrace{\begin{bmatrix} I & 0 & dx \\ 0 & I & dy \\ 0 & 0 & I \end{bmatrix}}_T \underbrace{\begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & I \end{bmatrix}}_R \underbrace{\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & I \end{bmatrix}}_S$$

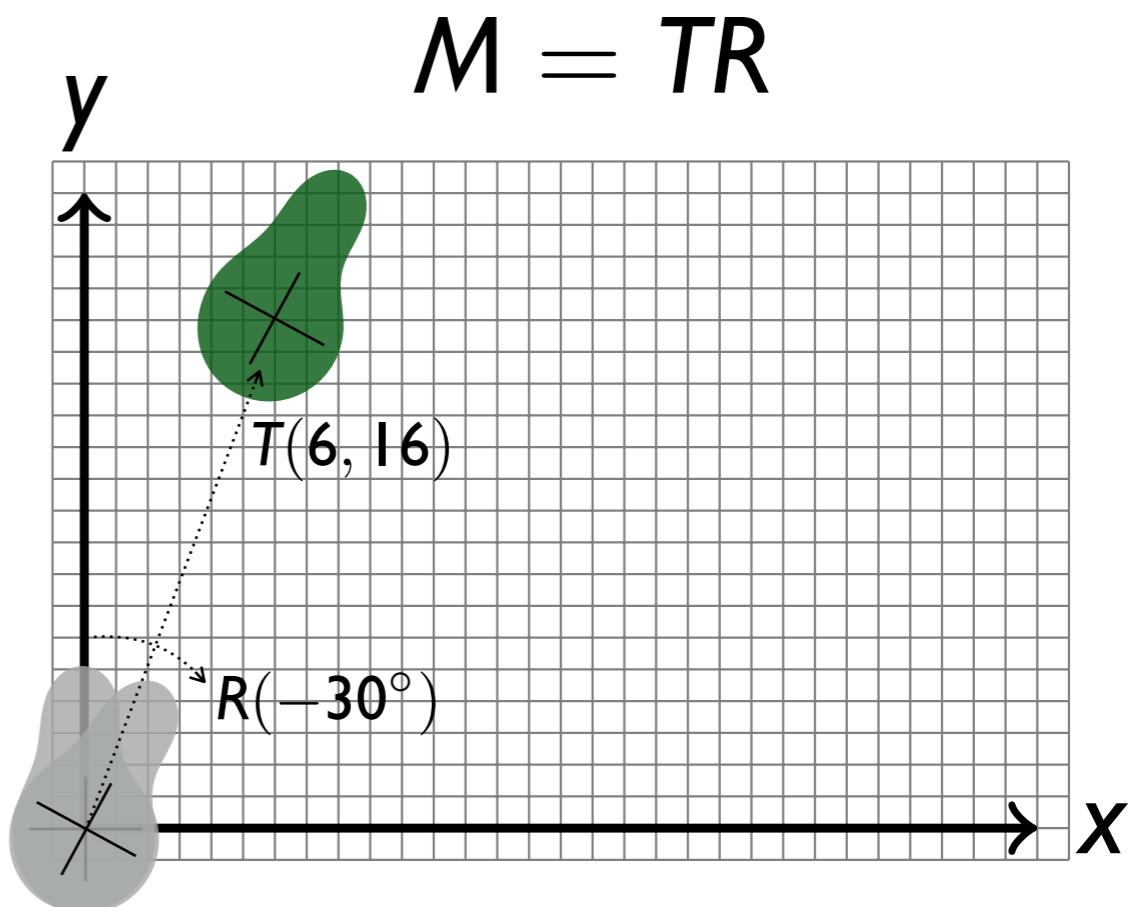
- Note: matrices are applied in sequence right to left.

# Composition of Transformations

- Order of transformations matters!
- Matrix multiplication is **NOT commutative**



$\neq$



# Transformation About Arbitrary Point

- Previous matrices rotate and scale about origin.
- Using composition we can build a scale and rotation about arbitrary point (or *pivot*)  $p$

$$R_p(\phi) = T(p_x, p_y) R(\phi) T(-p_x, -p_y)$$

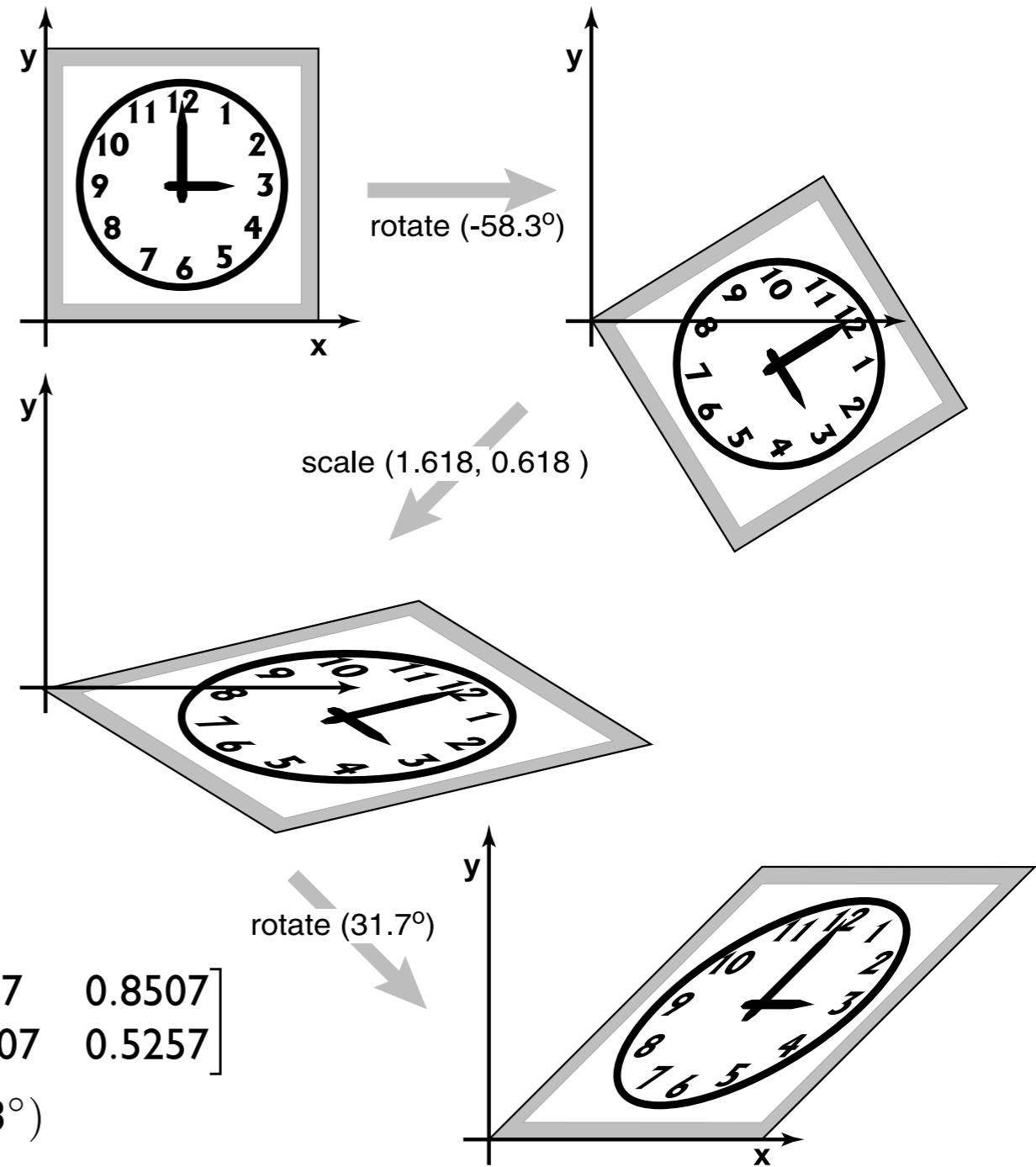
$$S_p(s_x, s_y) = T(p_x, p_y) S(s_x, s_y) T(-p_x, -p_y)$$

- In general, translate pivot to origin first, apply rotation/scaling, then translate back

# Decomposition of A Transformation

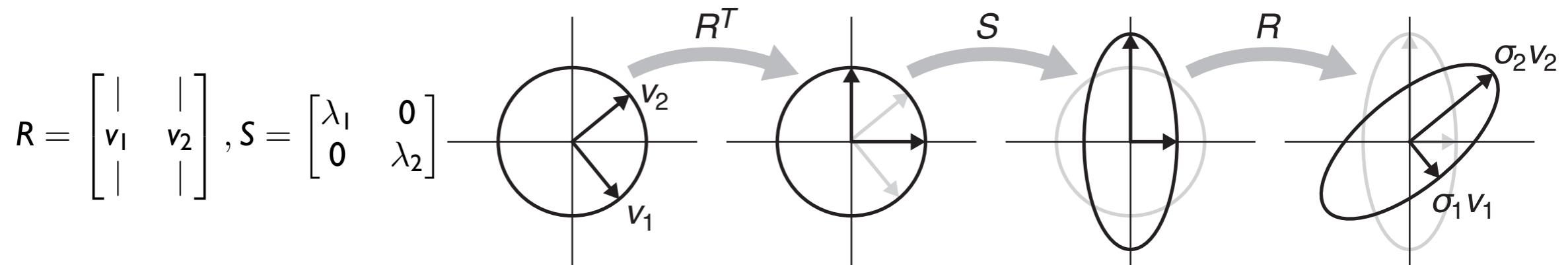
- “Undo” a composition of transformations
- Any 2D matrix can be decomposed into a product of rotation, scale, rotation
- E.g.: SVD of a shear matrix

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 1.618 & 0 \\ 0 & 1.618 \end{bmatrix} \begin{bmatrix} 0.5257 & 0.8507 \\ -0.8507 & 0.5257 \end{bmatrix}$$
$$= \text{rotate}(31.7^\circ) \text{scale}(1.618, 0.618) \text{rotate}(-58.3^\circ)$$



# Symmetric Eigenvalue Decomposition

- Eigenvalue decomposition:  $A = RSR^T$
- $A$ : symmetric,  $R$ : is orthogonal &  $S$ : is diagonal
- Interpretation: (simple scale in arbitrary direction)
  1. Rotate  $v_1$  and  $v_2$  to the X and Y axes (transform by  $R^T$ )
  2. Scale in x and y by  $(\lambda_1, \lambda_2)$  (transform by  $S$ )
  3. Rotate the X and Y axes back to  $v_1$  and  $v_2$  (transform by  $R$ )



# Symmetric Eigenvalue Decomposition

- Analytically reversing the diagonalization process

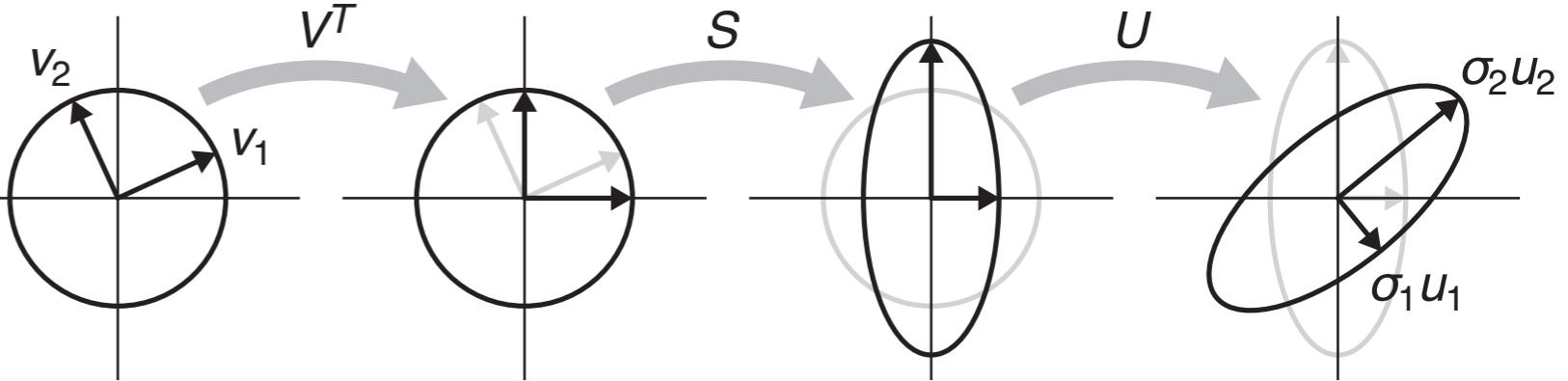
$$\begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} =$$
$$\begin{bmatrix} \lambda_1 \cos^2(\phi) + \lambda_2 \sin^2(\phi) & (\lambda_2 - \lambda_1) \cos(\phi) \sin(\phi) \\ (\lambda_2 - \lambda_1) \cos(\phi) \sin(\phi) & \lambda_2 \cos^2(\phi) + \lambda_1 \sin^2(\phi) \end{bmatrix}$$

- This tells us that, **symmetric matrices are just scaling operations** - albeit potentially nonuniform and non-axis aligned ones.

# Singular Value Decomposition (SVD)

- Singular Value Decomposition:  $A = USV^T$
- $A$ : non-symmetric,  $U$  and  $V$ : orthogonal &  $S$ : diagonal
- Interpretation:
  1. Rotate  $v_1$  and  $v_2$  to the X and Y axes (transform by  $V^T$ )
  2. Scale in x and y by  $(\sigma_1, \sigma_2)$  (transform by  $S$ )
  3. Rotate the X and Y axes to  $u_1$  and  $u_2$  (transform by  $U$ )

$$U = \begin{bmatrix} | & | \\ u_1 & u_2 \\ | & | \end{bmatrix}, V = \begin{bmatrix} | & | \\ v_1 & v_2 \\ | & | \end{bmatrix}, S = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}$$



# Paeth Decomposition

- Applies only to rotations
- Uses shears to represent non-zero rotations

$$\begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} = \begin{bmatrix} 1 & \frac{\cos \phi - 1}{\sin \phi} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \sin \phi & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{\cos \phi - 1}{\sin \phi} \\ 0 & 1 \end{bmatrix}$$

- Very useful in raster image rotations
  - Image shears can be implemented efficiently using pixel translations

$$\begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i + sj \\ j \end{bmatrix} \quad (i, j) \text{ is a pixel position}$$

# 3D Transformations

# 3D Transformations

Transformer	Matrix
Scaling	$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Rotation	$\underbrace{\begin{bmatrix} \cos \phi & -\sin \phi & 0 & 0 \\ \sin \phi & \cos \phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{rotate}_z(\phi)}, \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi & 0 \\ 0 & \sin \phi & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{rotate}_x(\phi)}, \underbrace{\begin{bmatrix} \cos \phi & 0 & \sin \phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \phi & 0 & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{rotate}_y(\phi)}$
Translation	$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$

# 3D Transformations

- Rotation is considerably more complicated in 3D than in 2D due to more possible axes of rotation
- Any 3D linear transformation matrix can be decomposed using SVD into rotation, scale and rotation
- Any symmetric 3D linear transformation matrix can be decomposed using Eigen decomposition into rotation, scale and inverse rotation
- A 3D rotation can be decomposed into a product of 3D shear matrices

# Arbitrary 3D Rotations

- Given three mutually orthogonal unit vectors

$$u = x_u \hat{i} + y_u \hat{j} + z_u \hat{k}$$

$$v = x_v \hat{i} + y_v \hat{j} + z_v \hat{k}$$

$$w = x_w \hat{i} + y_w \hat{j} + z_w \hat{k}$$

- $R_{uvw} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}$ , takes the  $uvw$  basis to corresponding cartesian basis via rotation.

# Arbitrary 3D Rotations

- In order to rotate about an arbitrary vector  $a$ :
  1. Form an orthonormal  $uvw$  basis with  $w = a$
  2. Rotate  $uvw$  basis to canonical basis  $xyz$
  3. Rotate about the z-axis by  $\phi$
  4. Rotate the canonical basis back to  $uvw$  basis

$$R_a(\phi) = \begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}$$

# Constructing a Basis from a Single Vector

- No unique basis, but we just need any one (robust)

I. Make  $w$  a unit vector:  $w = \frac{a}{\|a\|}$

2. Choose any vector  $t$  not collinear with  $w$ , and build  $u$  as:

$$u = \frac{t \times w}{\|t \times w\|}$$

3. Complete the basis by computing  $v$  as:

$$v = w \times u$$

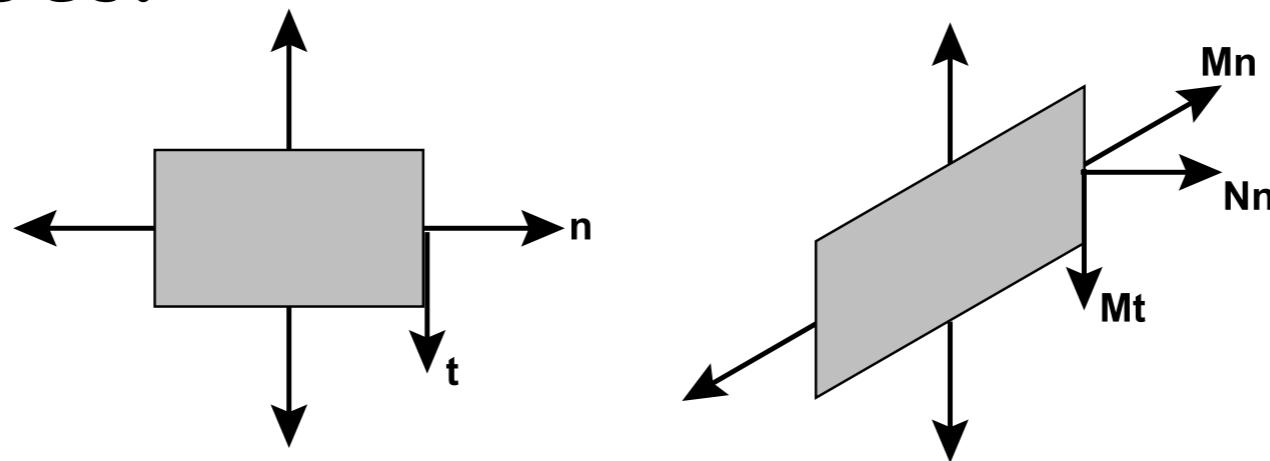
---

A simple procedure to find  $t$  significantly different from  $w$  is:

1. Start with  $t$  equal to  $w$
2. Change the smallest magnitude component of  $t$  to 1

# Transforming Normals

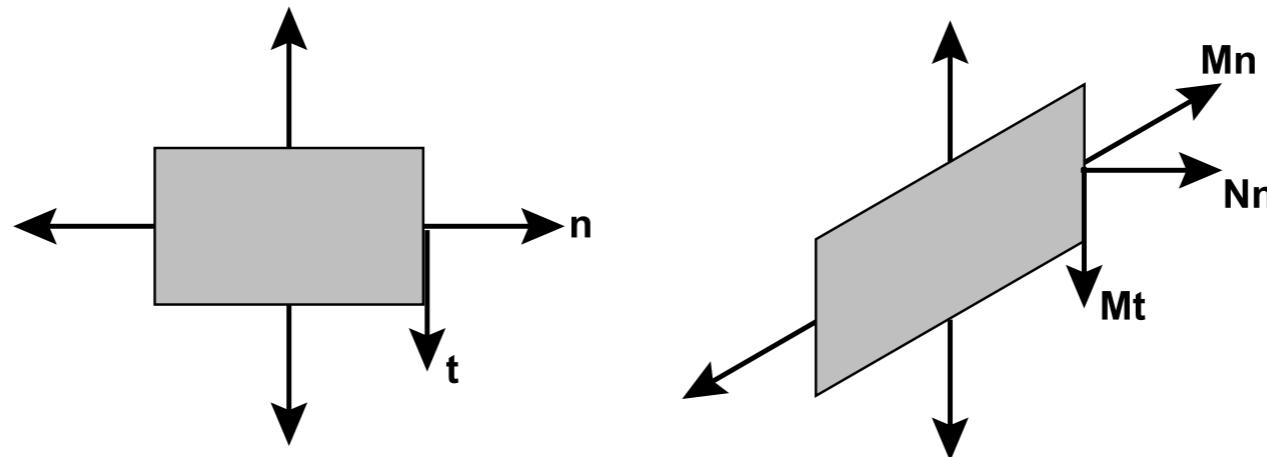
- Apart from representing positions, 3D vectors can also represent surface *normals*
- Surface normal vectors are orthogonal to the tangent plane of the surface at a point
- Unfortunately, these do not transform the way surface does!



# Transforming Normals

- Let the surface points be transformed by matrix  $M$ , and let  $N$  be the suitable matrix for the normal  $n$
- Tangent  $t$  and normal  $n$  are orthogonal:  $n^T t = 0$

$$N = (M^{-1})^T$$



# Reading

- FCG: 6

---

ICG: Interactive Computer Graphics, E. Angel, and D. Shreiner, 6th ed.

FCG: Fundamentals of Computer Graphics, P. Shirley, M. Ashikhmin, and S. Marschner, 3rd ed.