Computer Graphics
CS 543 – Lecture 4 (Part 3)
Introduction to Transformations (Part 2)

Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Introduction to Transformations

- Transformation changes an objects:
  - Position (translation)
  - Size (scaling)
  - Orientation (rotation)
  - Shapes (shear)
- Introduce first in 2D or *(x,y)*, build intuition
- Later, talk about 3D
- Transform object by applying sequence of matrix multiplications to object vertices

Transformations in OpenGL

- Pre 3.0 OpenGL had a set of transformation functions (now deprecated)
  - glTranslate( )
  - glRotate( )
  - glScale( )

Transformations in OpenGL

- OpenGL would previously receive transform commands, maintain concatenations of transform matrices as **modelview matrix**
- No longer
- Programmer ***may*** now choose to maintain modelview **or NOT!**

Transformations in OpenGL

- Three choices
  - Application code
  - GLSL functions
  - vec.h and mat.h

Why Matrices?

- All transformations can be performed using matrix/vector multiplication

- Allows pre-multiplication of all matrices

- Note: point (x,y) needs to be represented as (x,y,1), also called **Homogeneous coordinates**

Homogenous Coordinates

- Homogeneous coordinates representation of point

  P = (Px,Py,Pz) => (Px,Py,Pz,1)

- We could introduce arbitrary scaling factor, w, so that

  P = (wPx, wPy, wPz, w) (**Note:** w is non-zero)

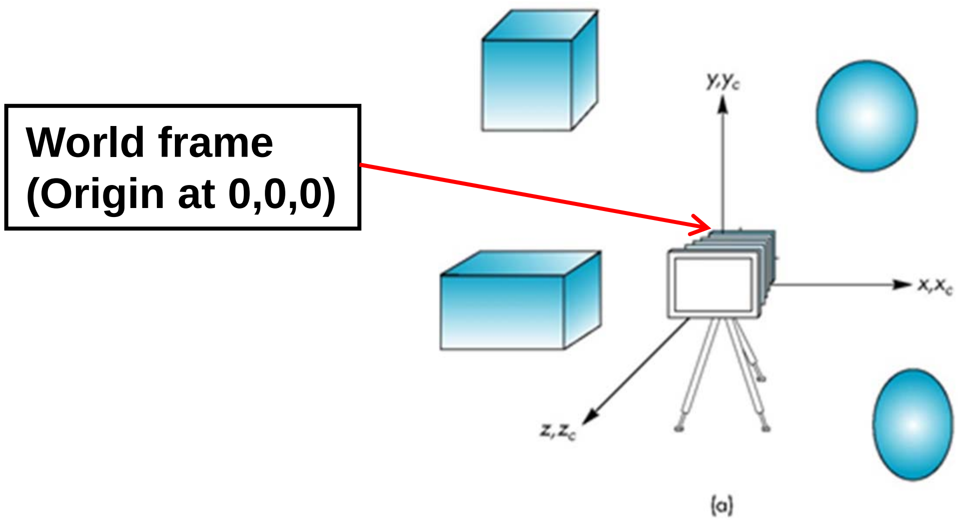
- For example, the point P = (2,4,6) can be expressed as
  - (2,4,6,1)
  - or (4,8,12,2) where w=2
  - or (6,12,18,3) where w = 3, or....
- To convert from homogeneous back to ordinary coordinates, first divide all four terms by **w** and discard 4^{th} term

Homogeneous Coordinates and Computer Graphics

- Homogeneous coordinates are key in graphics
  - Transformations (rotation, translation, scaling) can be implemented with matrix multiplications using 4 x 4 matrices
  - Hardware pipeline works with 4 dimensional representations

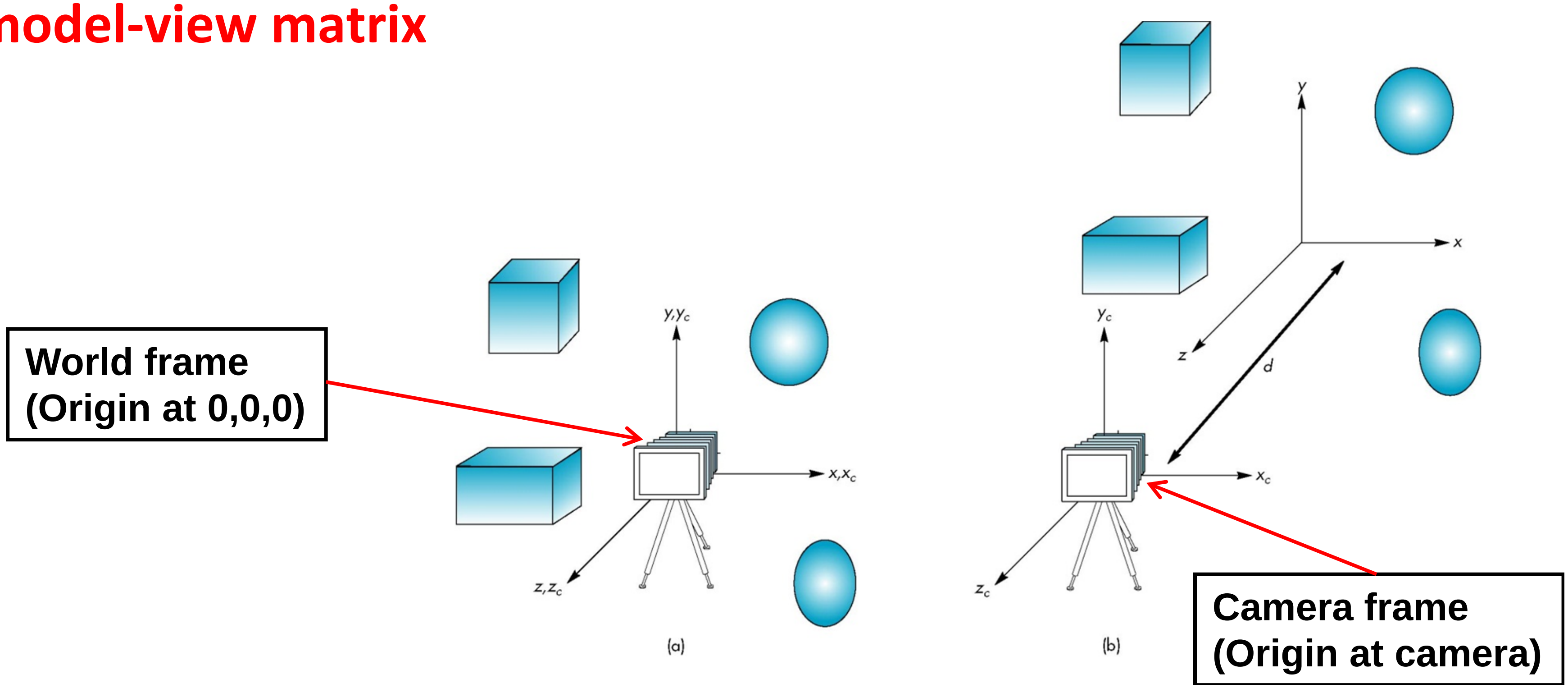
The World Frames

- In OpenGL, objects/scene initially defined in **world frame**
- Transformations (translate, scale, rotate) applied to objects in **world frame**



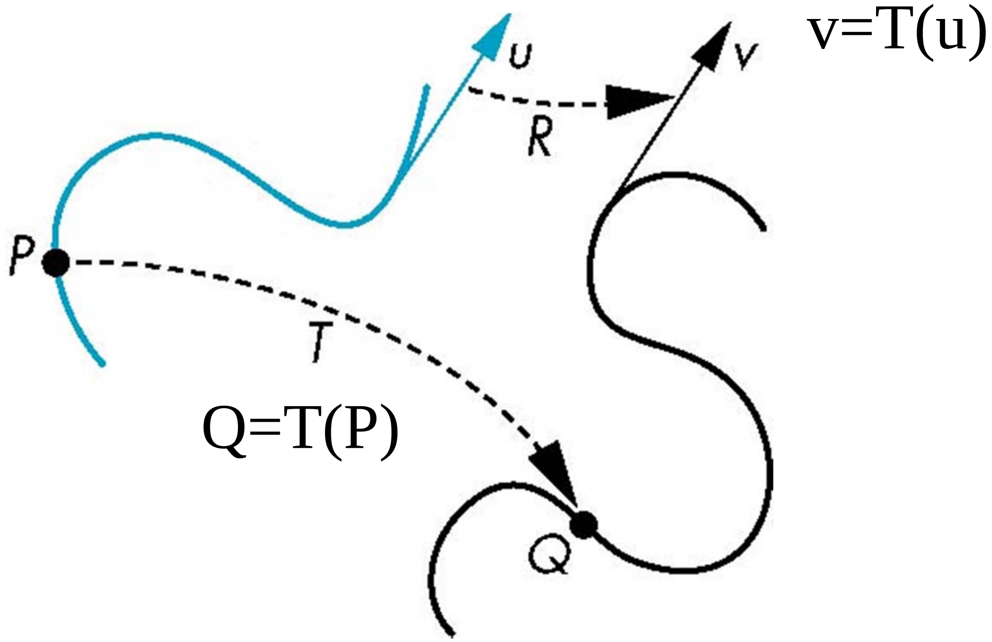
Camera Frame

- After we define a camera (eye) position
- We then represent objects in **camera frame** (origin at eye position)
- objects moved from world frame to camera frame using **model-view matrix**



General Transformations

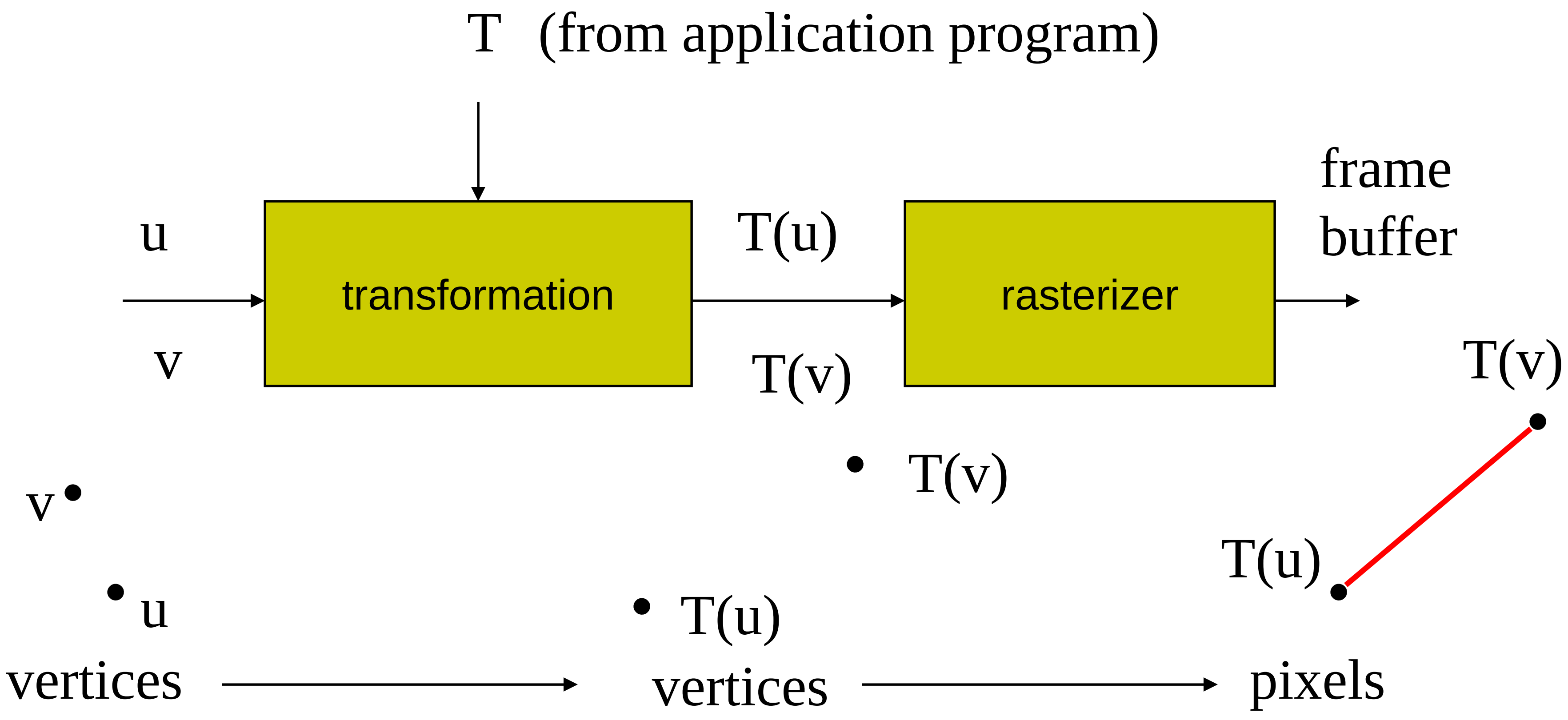
A transformation maps points to other points and/or vectors to other vectors



Affine Transformations

- **Rigid body transformations:** rotation, translation, scaling, shear
- Line preserving: important in graphics since we can
  1. Transform endpoints of line segments
  2. Draw line segment between the transformed endpoints

Pipeline Implementation



Point Representation

- We use a column matrix (2x1 matrix) to represent a 2D point

$$\begin{pmatrix} x \\ y \end{pmatrix}$$

- General form of transformation of a point *(x,y)* to *(x',y')* can be written as:

$$x' = ax + by + c$$

$$y' = dx + ey + f$$

or

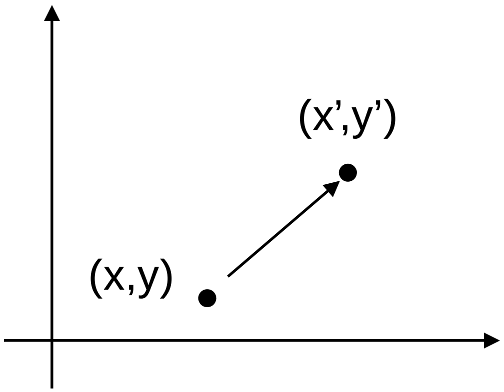
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \bullet \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Translation

- To reposition a point along a straight line
- Given point *(x,y)* and translation distance (t_x, t_y)
- The new point: *(x',y')*

$x' = x + t_x$

$y' = y + t_y$



or

$$P' = P + T \quad \text{where} \quad P' = \begin{pmatrix} x' \\ y' \end{pmatrix} \quad P = \begin{pmatrix} x \\ y \end{pmatrix} \quad T = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

3x3 2D Translation Matrix

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

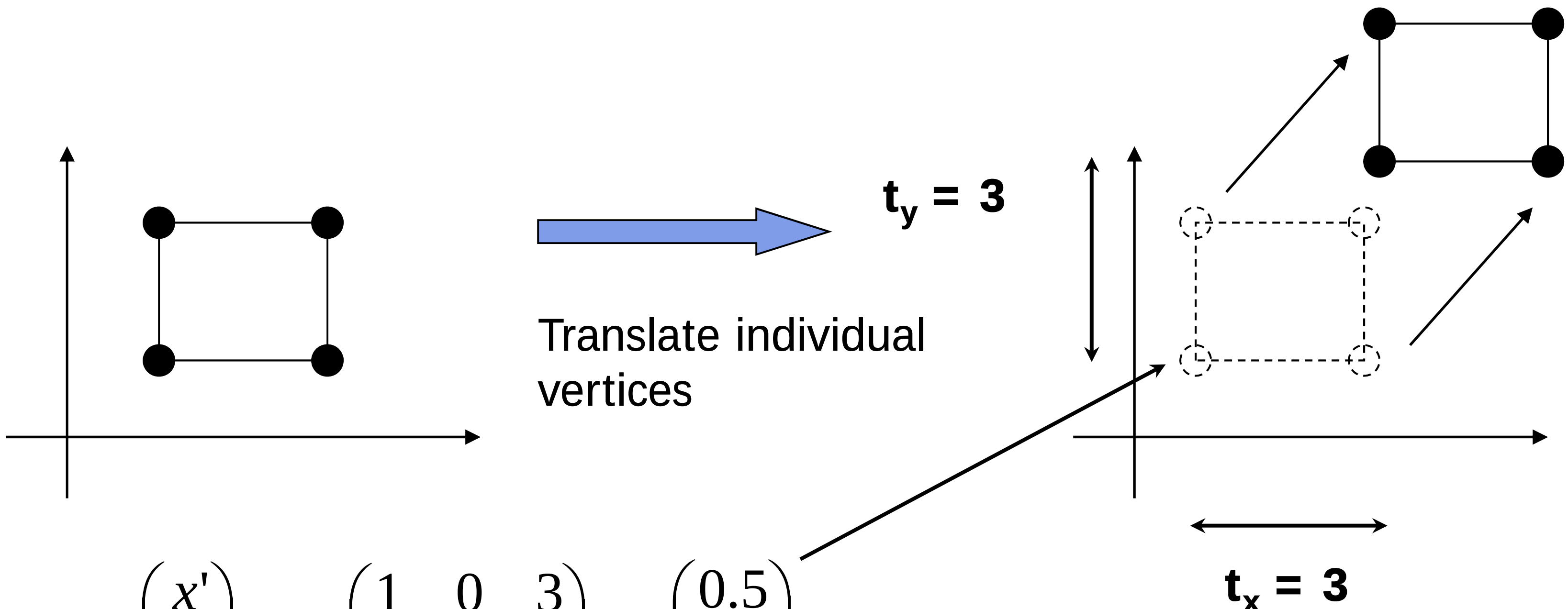
use 3x1 vector

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Note: it becomes a matrix-vector multiplication

2D Translation of Objects

- How to translate an object with multiple vertices?

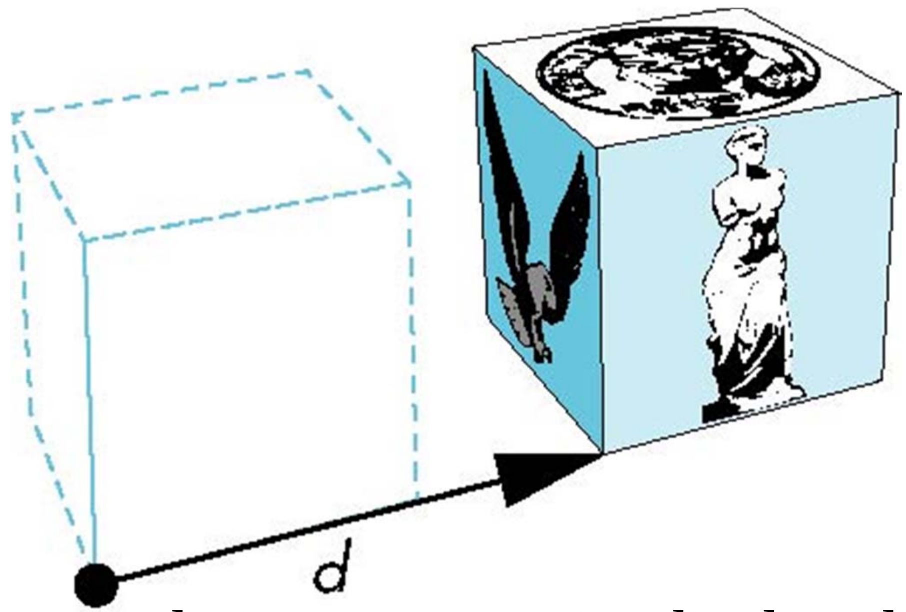


$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 0.5 \\ 0.5 \\ 1 \end{pmatrix}$$

3D Translation

- Move each vertex by same distance **d = (d_x, d_y, d_z)**

object



translation: every point displaced by same vector

Transforms in 3D

- 2D: 3x3 matrix multiplication
- 3D: 4x4 matrix multiplication: homogenous coordinates
- Again: transform object = transform each vertice
- General form:

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Xform of *P* →

$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = M \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

3D Translation Matrix

- Now, 3D :

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

translate(tx,ty,tz)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

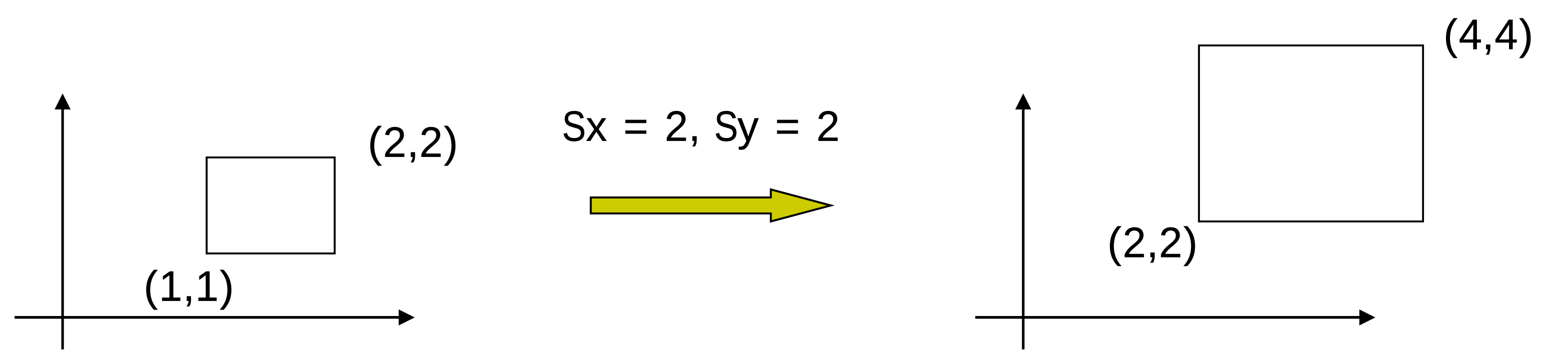
- Where: *x'= x.1 + y.0 + z.0 + tx.1 = x + tx*, … etc

2D Scaling

- Scale: Alter object size by scaling factor (s_x, s_y). i.e

$$x' = x \cdot Sx$$
$$y' = y \cdot Sy$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} Sx & 0 \\ 0 & Sy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



2D Scaling Matrix

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} Sx & 0 \\ 0 & Sy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

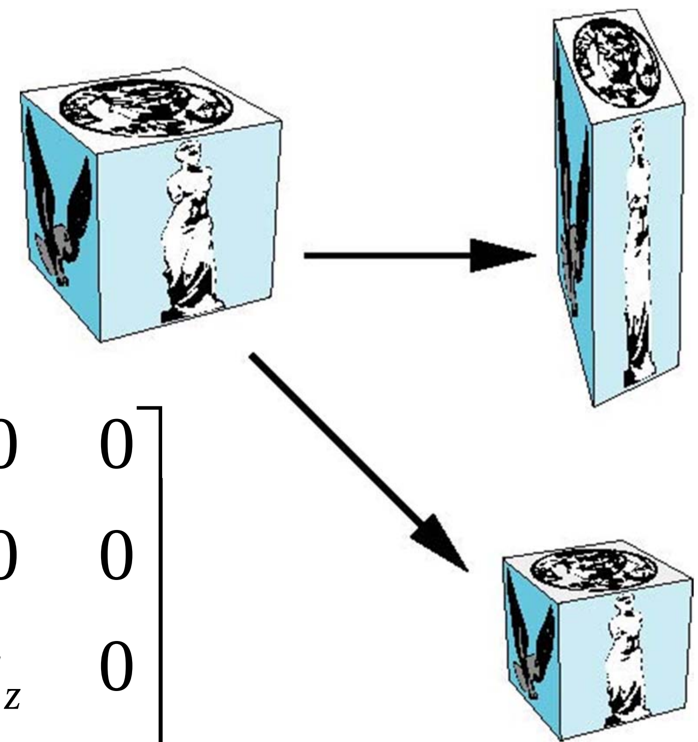
Scaling

Expand or contract along each axis (fixed point of origin)

$$x' = s_x x$$
$$y' = s_y x$$
$$z' = s_z x$$

$$\mathbf{p}' = \mathbf{S}\mathbf{p}$$

$$\mathbf{S} = \mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



4x4 3D Scaling Matrix

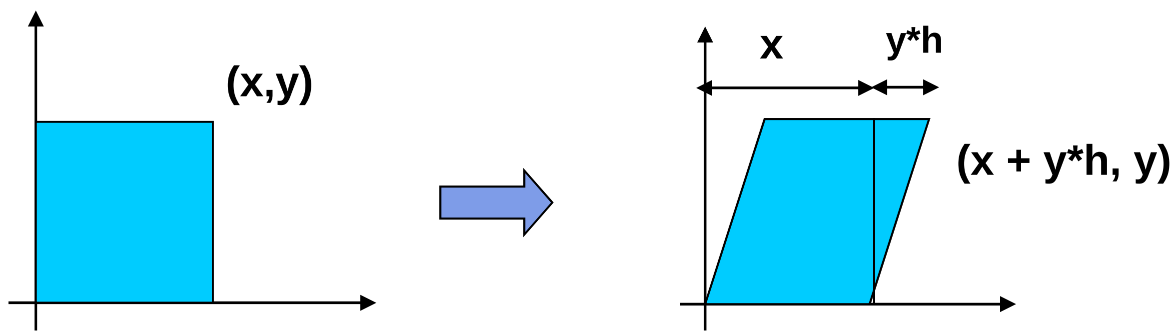
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Scale(Sx,Sy,Sz)

- Example:
- If $Sx = Sy = Sz = 0.5$
- Can scale:
- big cube (sides = 1) to small cube (sides = 0.5)
- 2D: square, 3D cube

Shearing

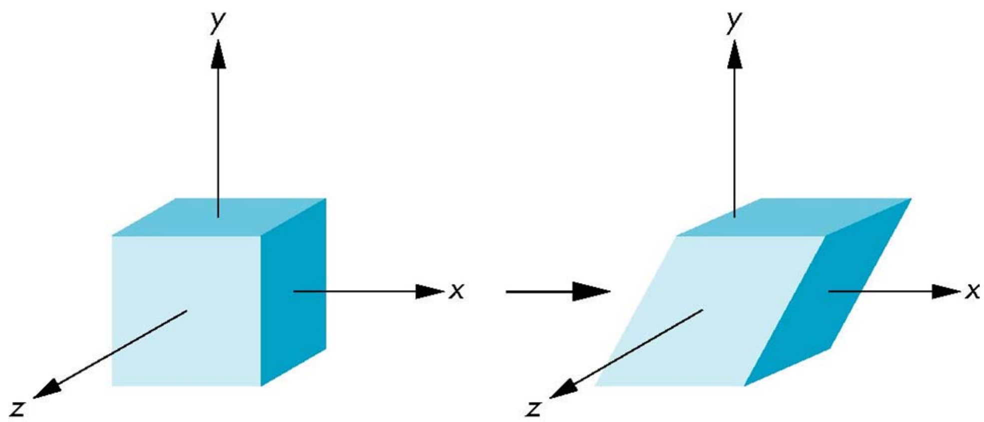


- Y coordinates are unaffected, but x cordinates are translated linearly with y
- That is:
  - **y' = y**
  - **x' = x + y * h**

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

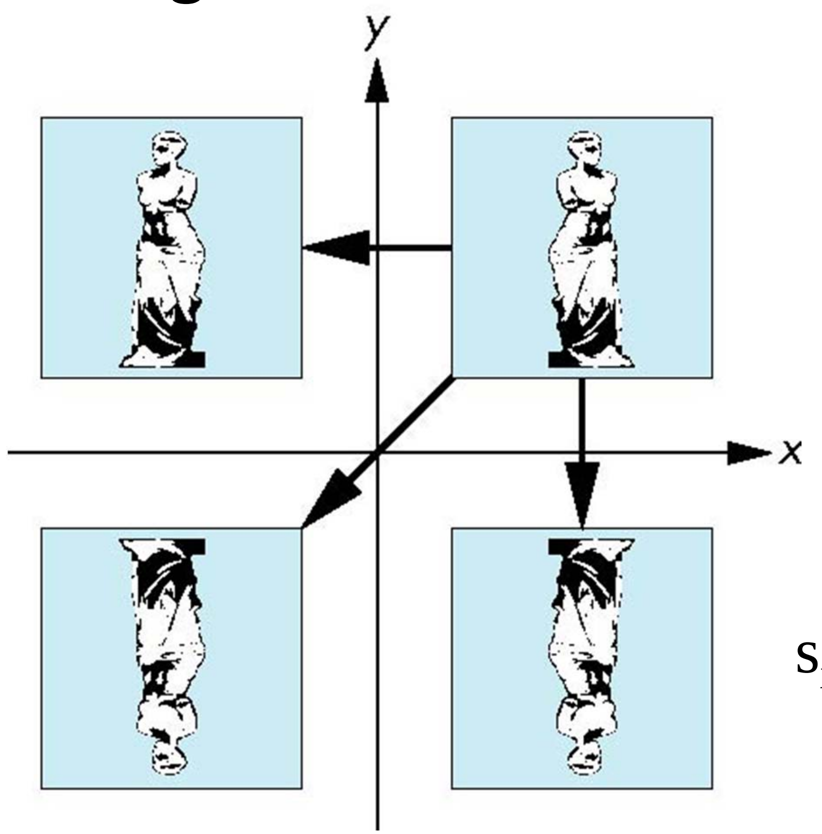
- h is fraction of y to be added to x

3D Shear



Reflection

corresponds to negative scale factors



$s_x = -1\ s_y = 1$

original

$s_x = -1\ s_y = -1$

$s_x = 1\ s_y = -1$

References

- Angel and Shreiner
- Hill and Kelley