Computer Graphics (CS 543)
Lecture 4a: Introduction to Transformations

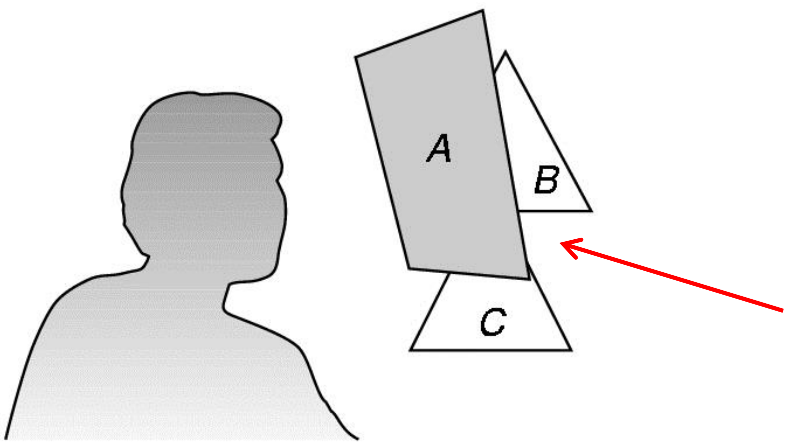
Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Hidden-Surface Removal

- If multiple surfaces overlap, we want to see only **closest**
- OpenGL uses *hidden-surface* technique called the ***z-buffer*** algorithm
- Z-buffer compares objects distances from viewer (depth) to determine closer objects



If overlap,
Draw face **A** (front face)
Do not draw faces **B** and **C**

Using OpenGL's *z*-buffer algorithm

- Z-buffer uses an extra buffer, (the z-buffer), to store depth information, compare distance from viewer
- 3 steps to set up Z-buffer:
  1. In **`main( )`** function

     ```
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)
     ```

  2. Enabled in **`init( )`** function

     ```
     glEnable(GL_DEPTH_TEST)
     ```

  3. Clear depth buffer whenever we clear screen

     ```
     glClear(GL_COLOR_BUFFER_BIT | DEPTH_BUFFER_BIT)
     ```

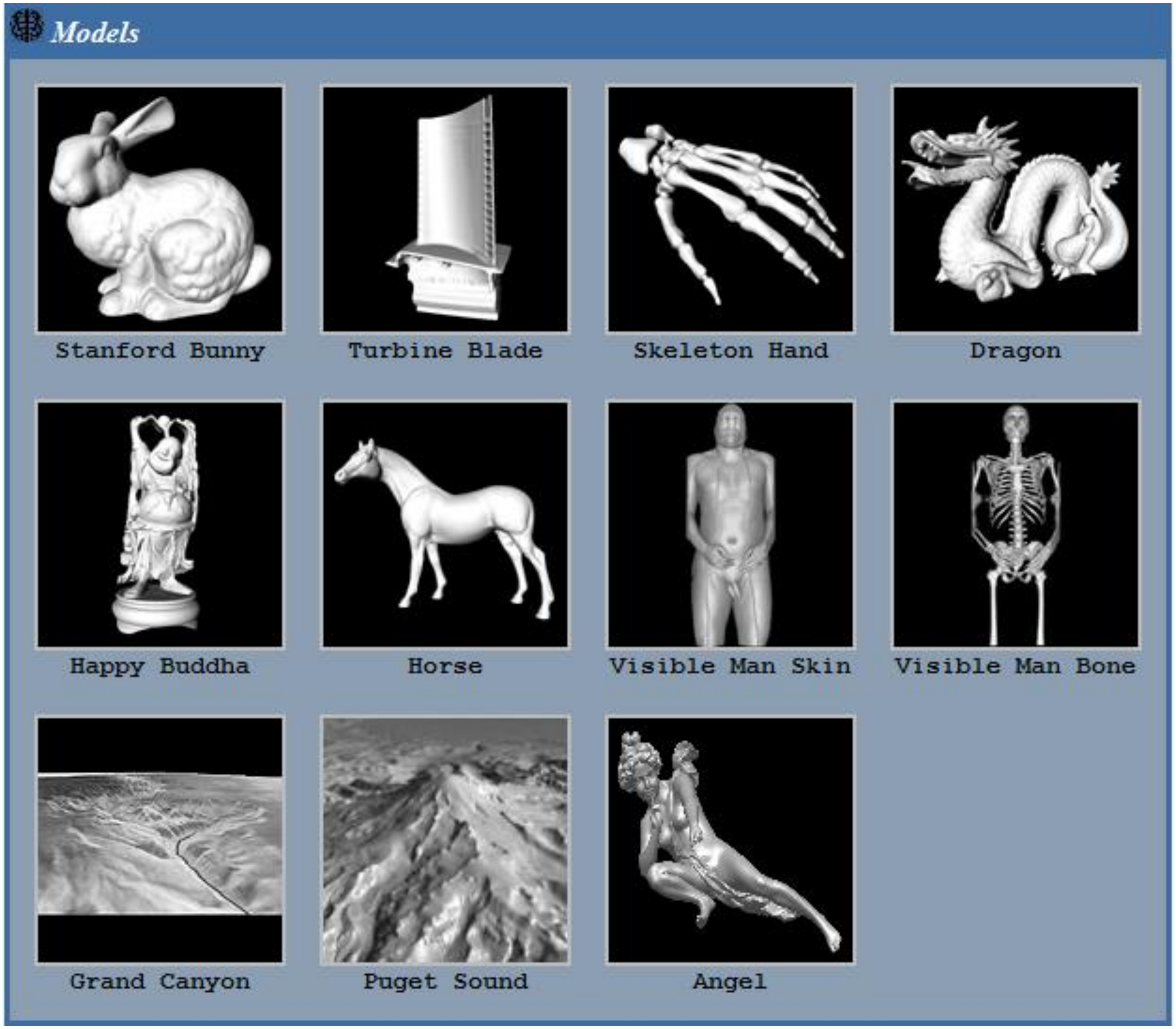
3D Mesh file formats

- 3D meshes usually stored in 3D file format
- Format defines how vertices, edges, and faces are declared
- Over 400 different file formats
- **Polygon File Format (PLY)** used a lot in graphics
- Originally PLY was used to store 3D files from 3D scanner
- We will use PLY files in this class

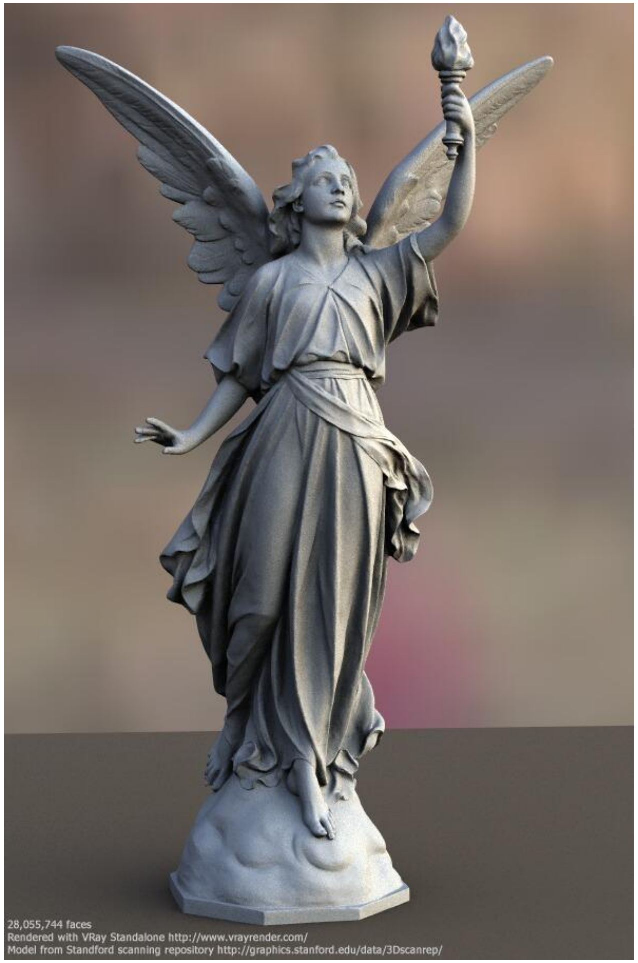
Sample PLY File

```
ply
format ascii 1.0
comment this is a simple file
obj_info any data, in one line of free form text element vertex 3
property float x
property float y
property float z
element face 1
property list uchar int vertex_indices
end_header
-1 0 0
0 1 0
1 0 0
3 0 1 2
```

Georgia Tech Large Models Archive



Stanford 3D Scanning Repository



Lucy: 28 million faces

Happy Buddha: 9 million faces

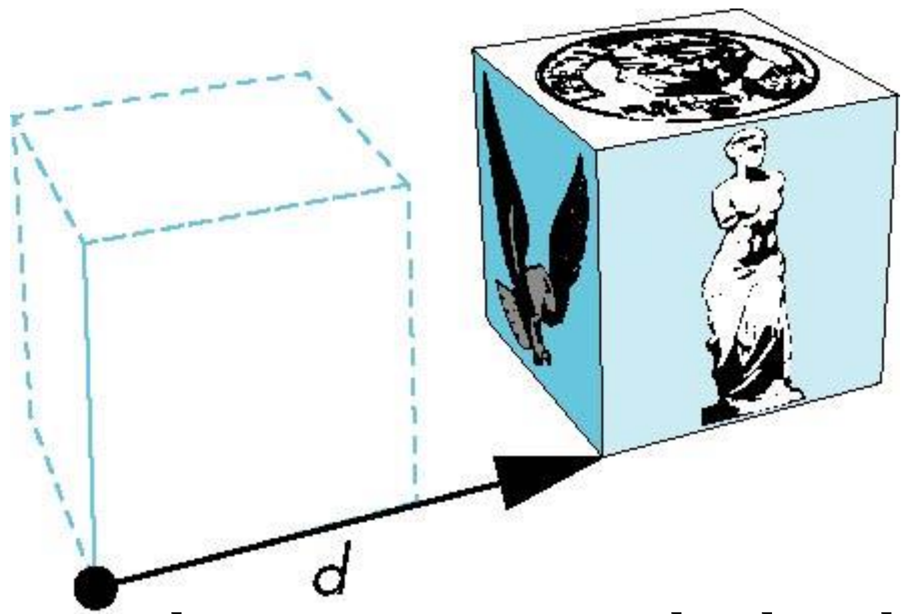
Introduction to Transformations

- May also want to transform objects by changing its:
  - Position (translation)
  - Size (scaling)
  - Orientation (rotation)
  - Shapes (shear)

Translation

- Move each vertex by **same** distance $\mathbf{d} = (\mathbf{d_x}, \mathbf{d_y}, \mathbf{d_z})$

object



translation: every point displaced by same vector

Scaling

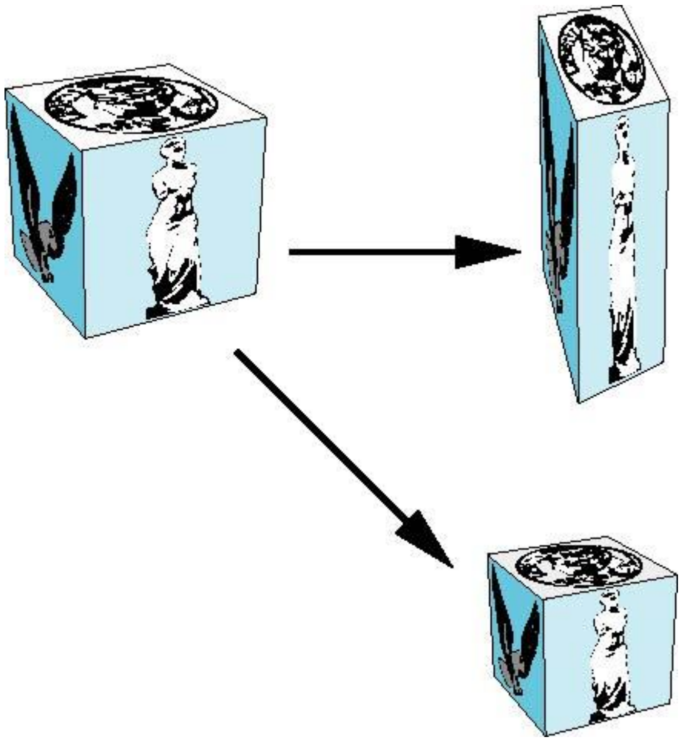
Expand or contract along each axis (fixed point of origin)

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

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

where

$$\mathbf{S} = \mathbf{S}(s_x, s_y, s_z)$$



Introduction to Transformations

- We can transform (translation, scaling, rotation, shearing, etc) object by applying matrix multiplications to object vertices

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

Original Vertex

Transformed Vertex

Transform Matrix

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

Why Matrices?

- Multiple transform matrices can be pre-multiplied
- One final resulting matrix applied (efficient!)
- For example:

transform 1 x **transform 2** x **transform 3** ....

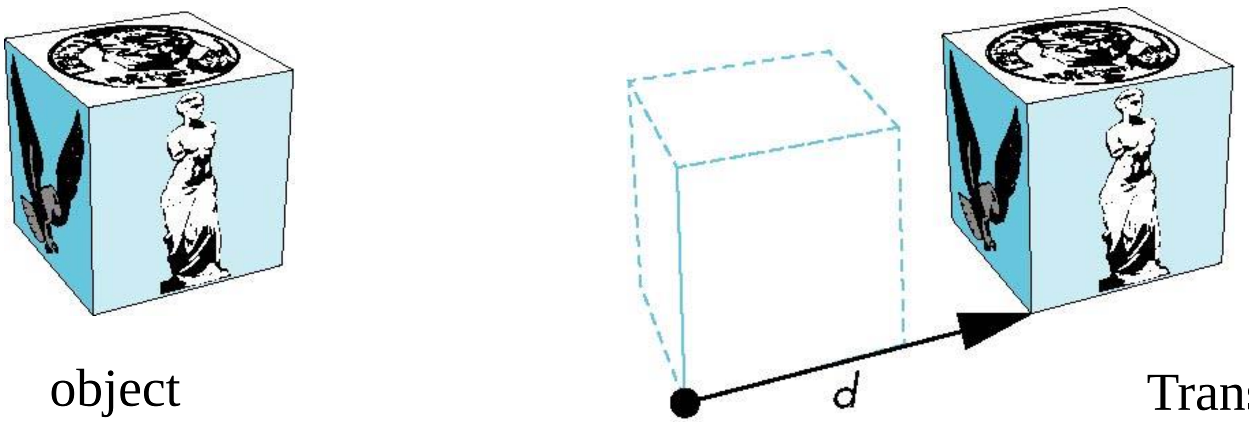
$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \boxed{\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

Transformed Point

Transform Matrices can
Be pre-multiplied

Original Point

3D Translation Example



object

Translation of object

- **Example:** If we translate a point (2,2,2) by displacement (2,4,6), new location of point is (4,6,8)

Using matrix multiplication for translation

Translate(2,4,6)

- Translate x: 2 + 2 = 4
- Translate y: 2 + 4 = 6
- Translate z: 2 + 6 = 4

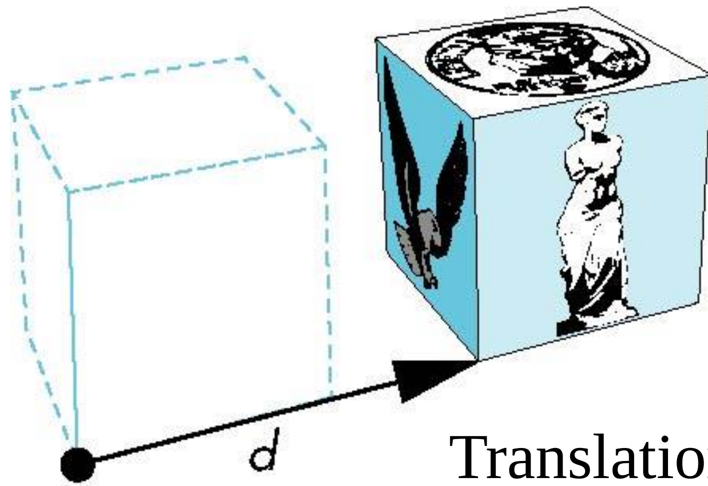
$$\begin{pmatrix} 4 \\ 6 \\ 8 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 2 \\ 2 \\ 2 \\ 1 \end{pmatrix}$$

Translated point **Translation Matrix** **Original point**

3D Translation

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

object



Translation of object

Translate(dx,dy,dz)

- Where:
  - *x'= x + dx*
  - *y'= y + dy*
  - *z'= z + dz*

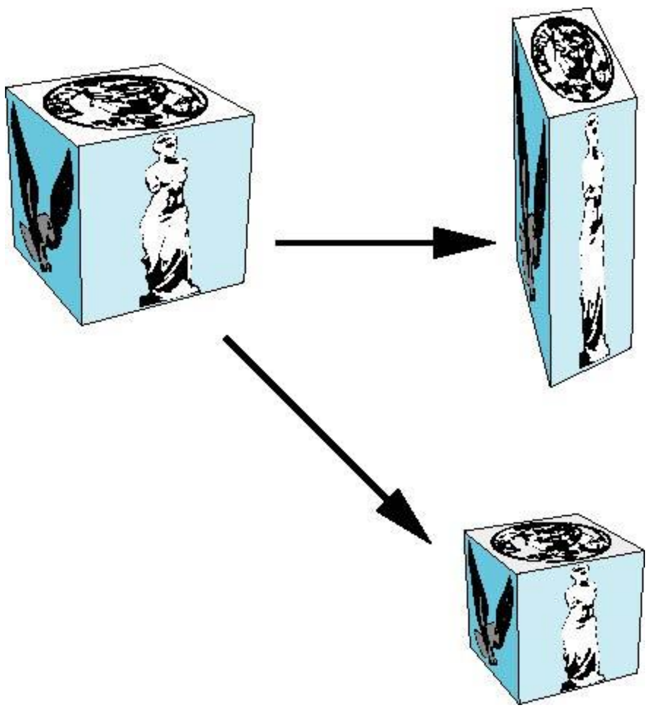
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Translation Matrix

Scaling Example

If we scale a point (2,4,6) by scaling factor (0.5,0.5,0.5)
Scaled point position = (1, 2, 3)

- Scale x: 2 x 0.5 = 1
- Scale y: 4 x 0.5 = 2
- Scale z: 6 x 0.5 = 3



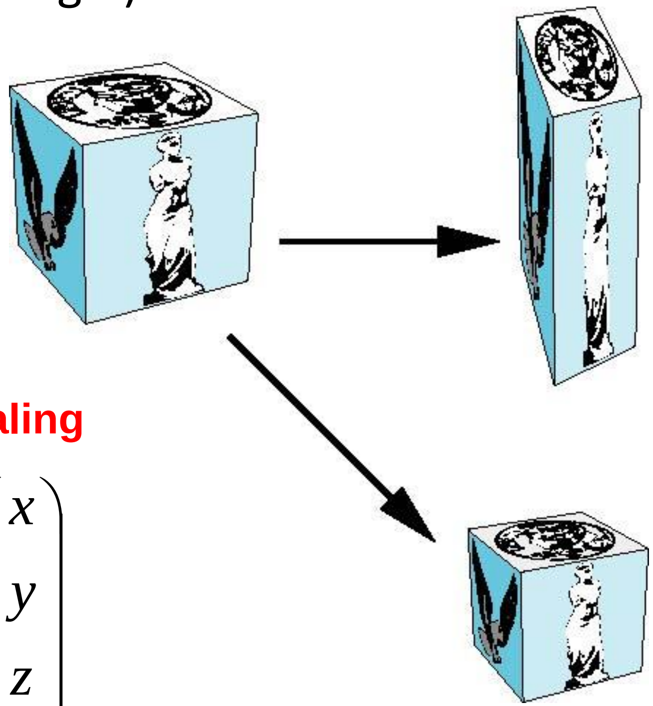
$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 2 \\ 4 \\ 6 \\ 1 \end{pmatrix}$$

Scale Matrix for Scale(0.5, 0.5, 0.5)

Scaling

Scale object = Move each object vertex by scale factor **S = (S_x, S_y, S_z)**
Expand or contract along each axis (relative to origin)

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



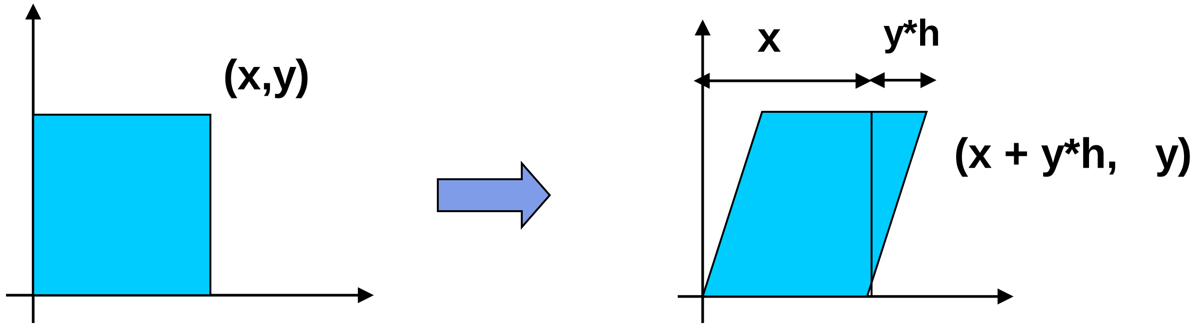
Using matrix multiplication for scaling

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

Scale Matrix

Scale(Sx,Sy,Sz)

Shearing

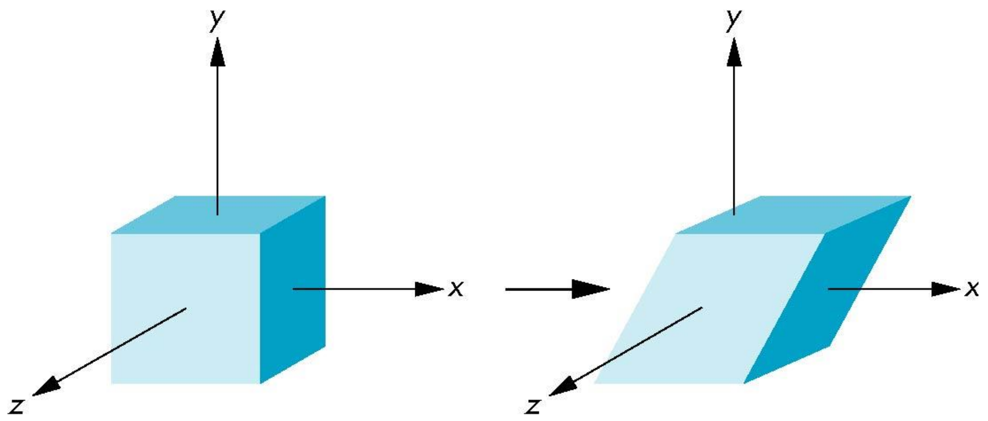


- Y coordinates are unaffected, but x cordinates are translated linearly with y
- That is:
  - **y’ = y**
  - **x’ = x + y * h**

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

- h is fraction of y to be added to x

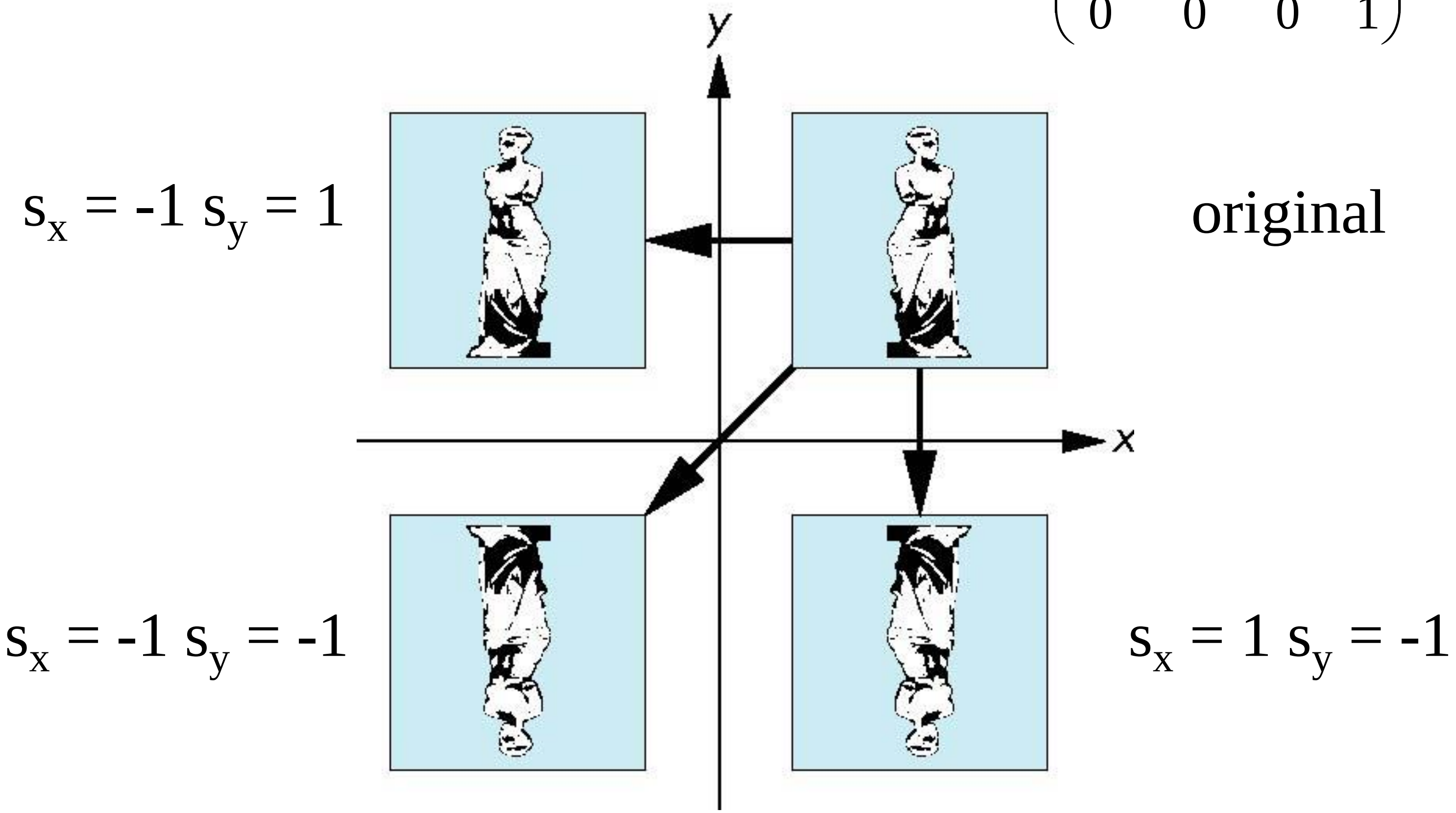
3D Shear



Reflection

- corresponds to negative scale factors

$$\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



References

- Angel and Shreiner, Chapter 3
- Hill and Kelley, Chapter 5