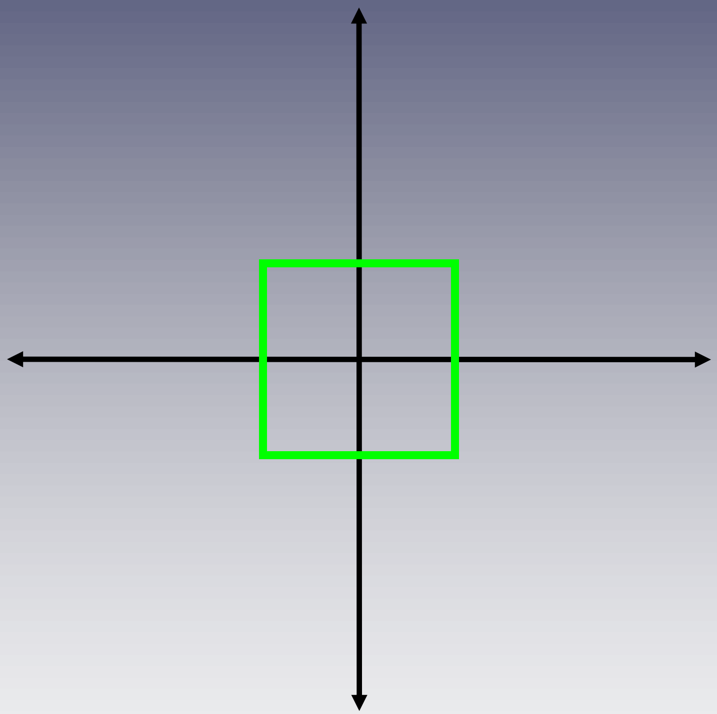
Geometry: 2-D & 3-D Transformations

Course web page:
http://goo.gl/EB3aA

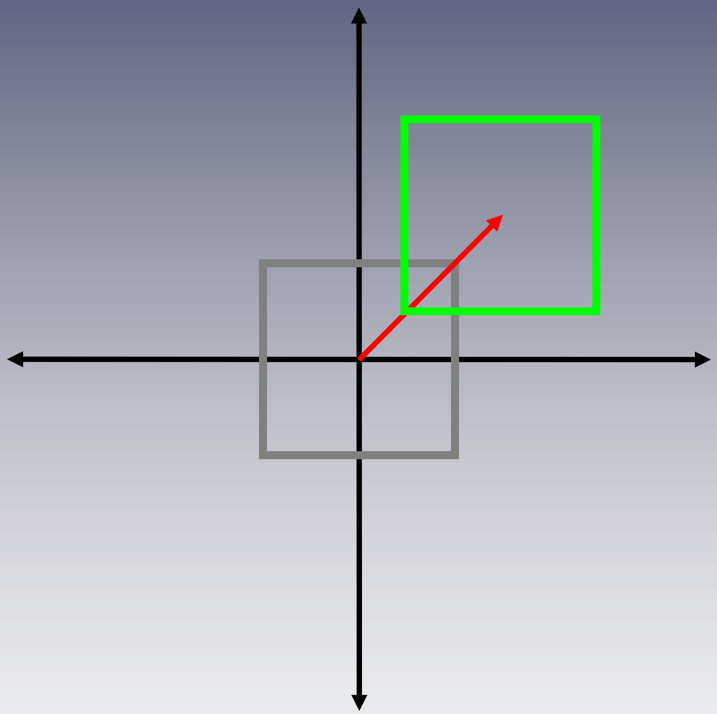
Outline

- HW #2 questions?
- 2-D, 3-D transformations
  - Types, mathematical representation
  - OpenGL functions for applying

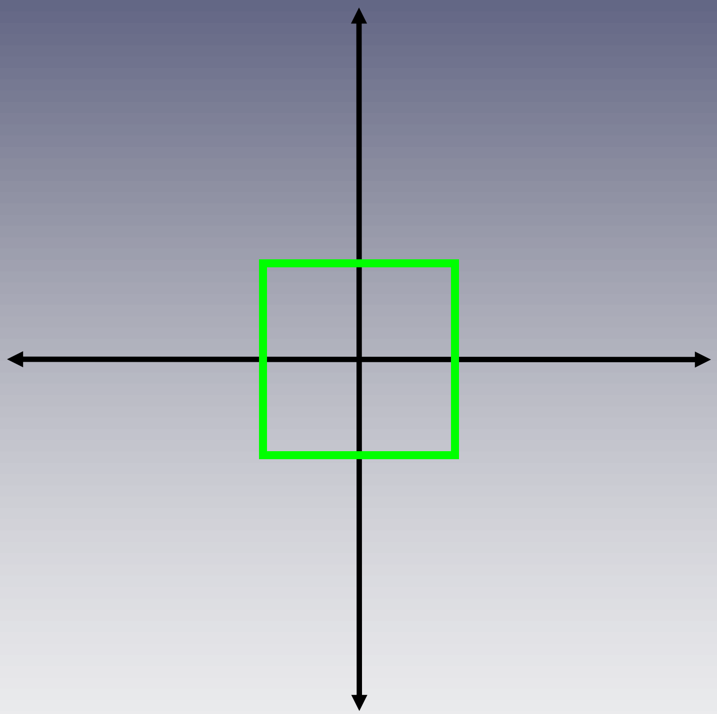
2-D Translation



2-D Translation

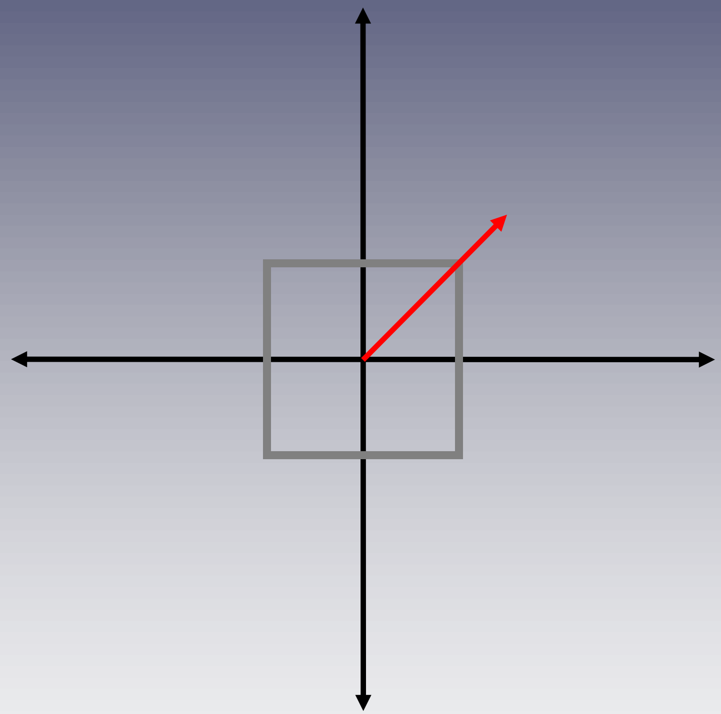


2-D Translation



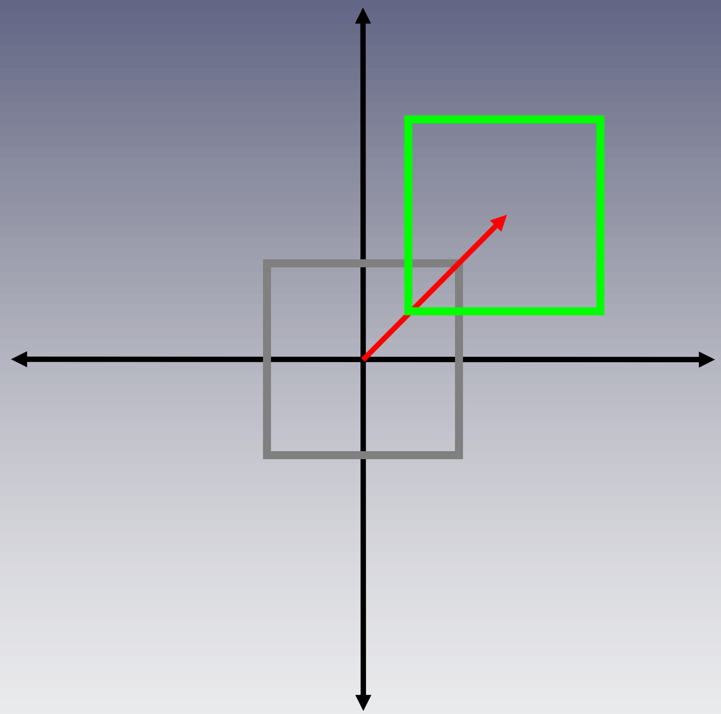
$$\begin{pmatrix} x \\ y \end{pmatrix}$$

2-D Translation



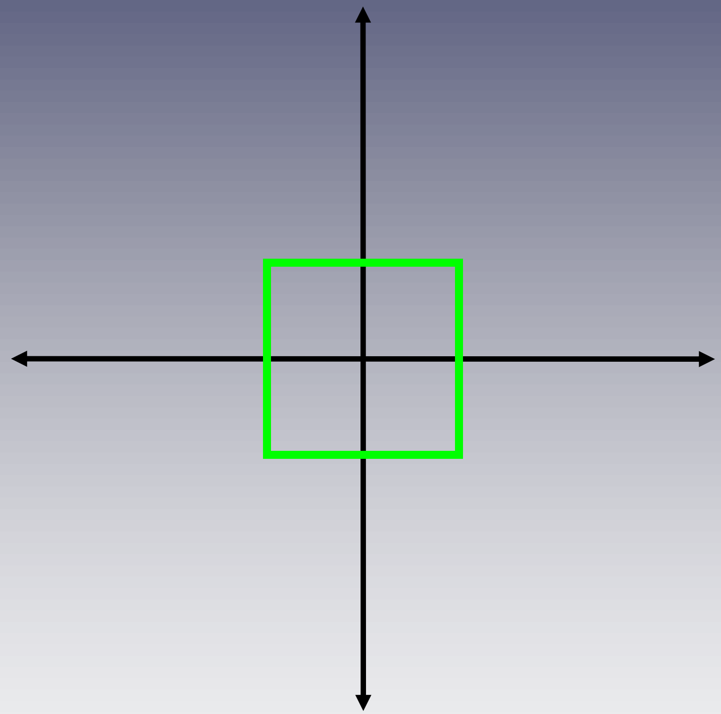
$$\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

2-D Translation

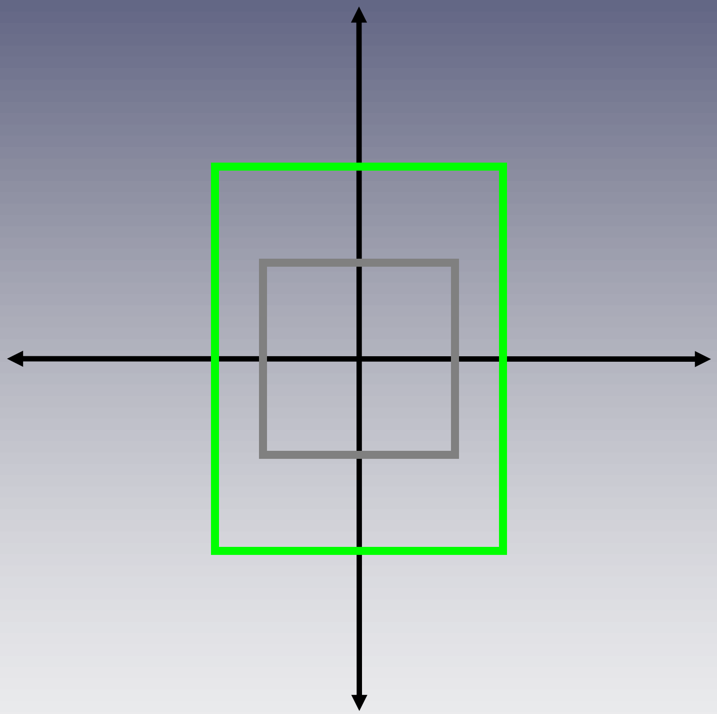


$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

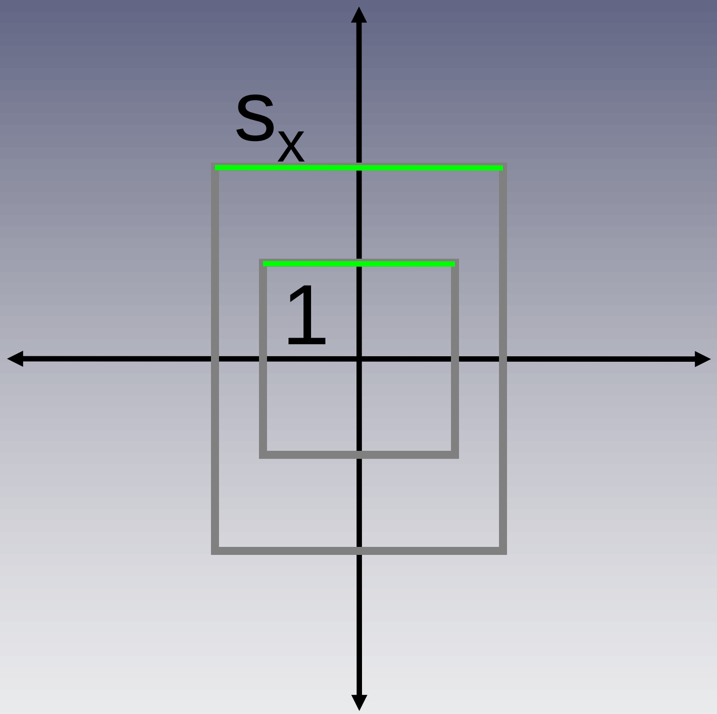
2-D Scaling



2-D Scaling

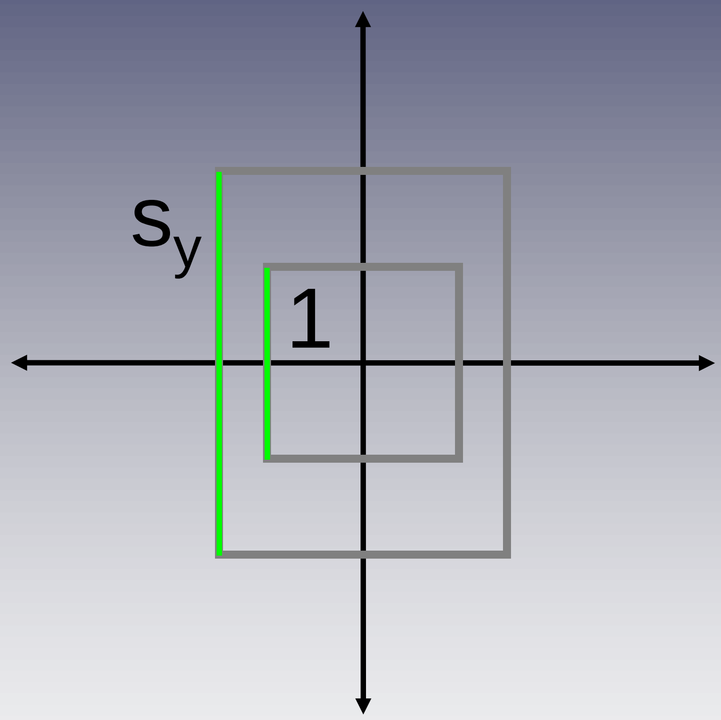


2-D Scaling



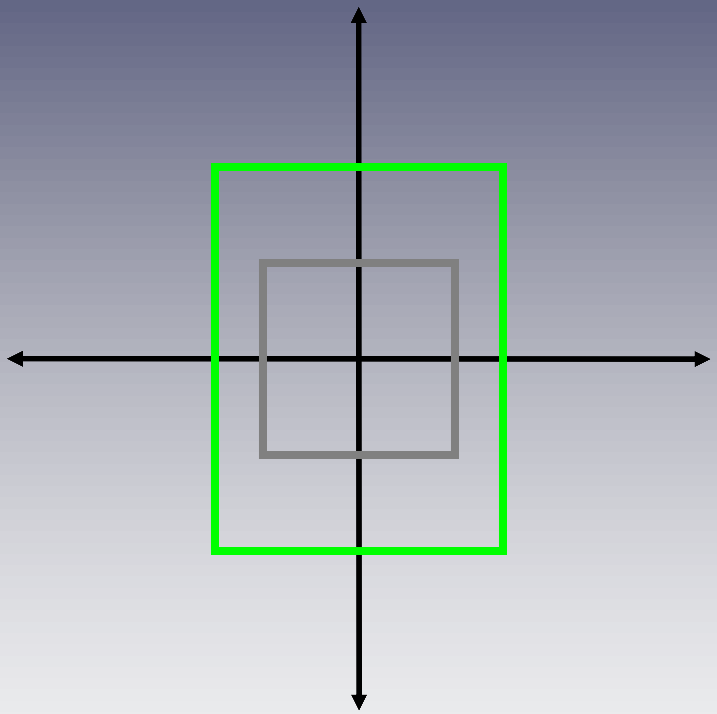
Horizontal shift proportional to horizontal position

2-D Scaling



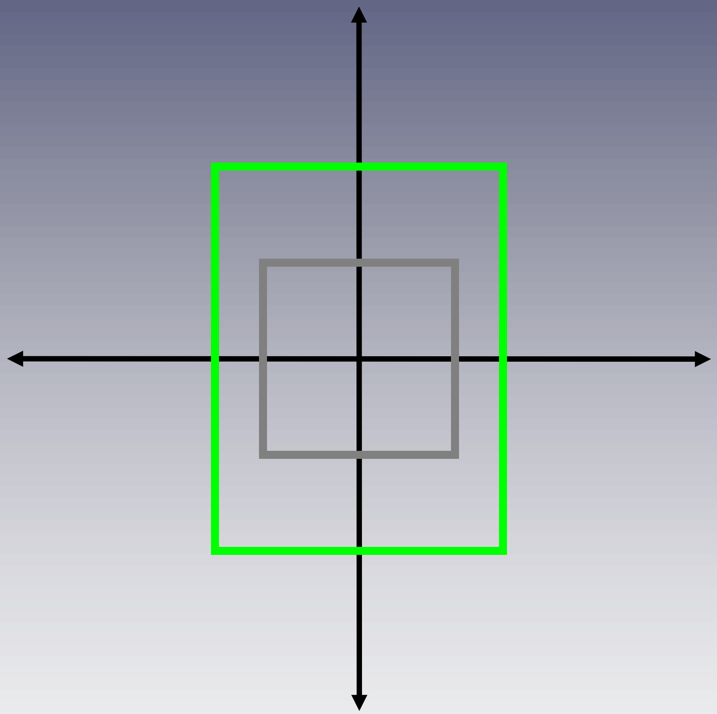
Vertical shift proportional to vertical position

2-D Scaling



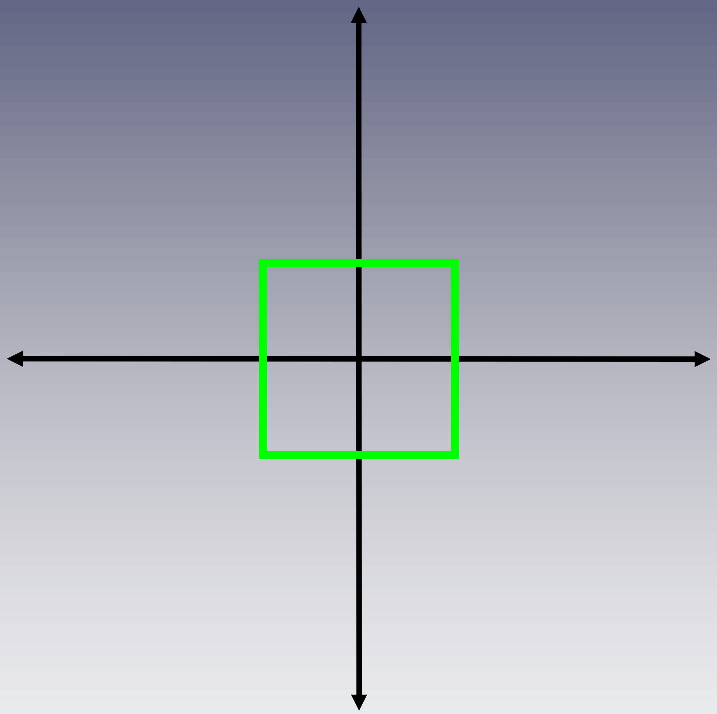
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x \cdot x \\ s_y \cdot y \end{pmatrix}$$

2-D Scaling

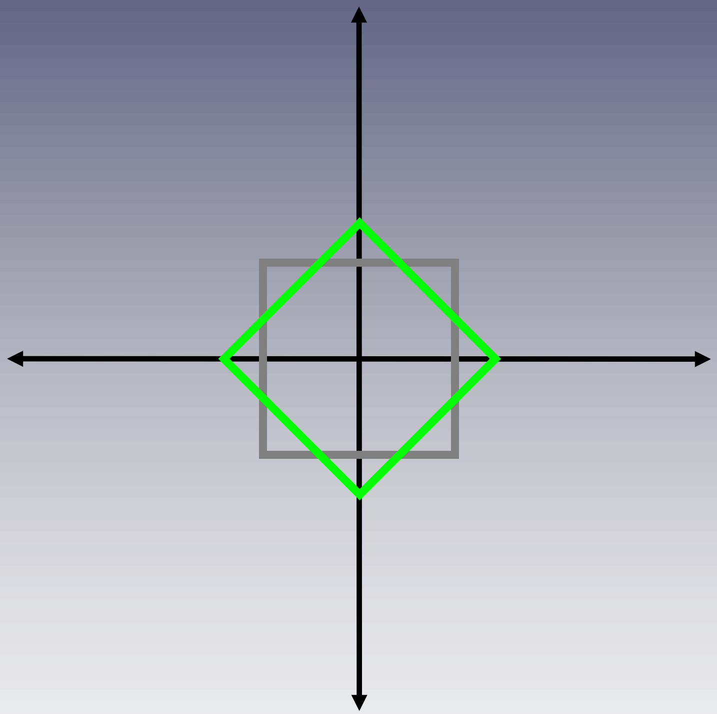


$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

2-D Rotation

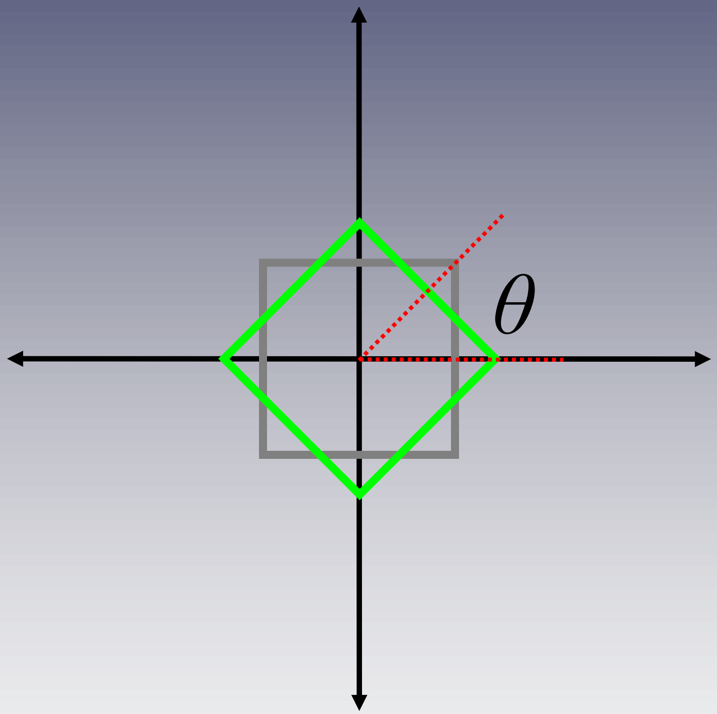


2-D Rotation



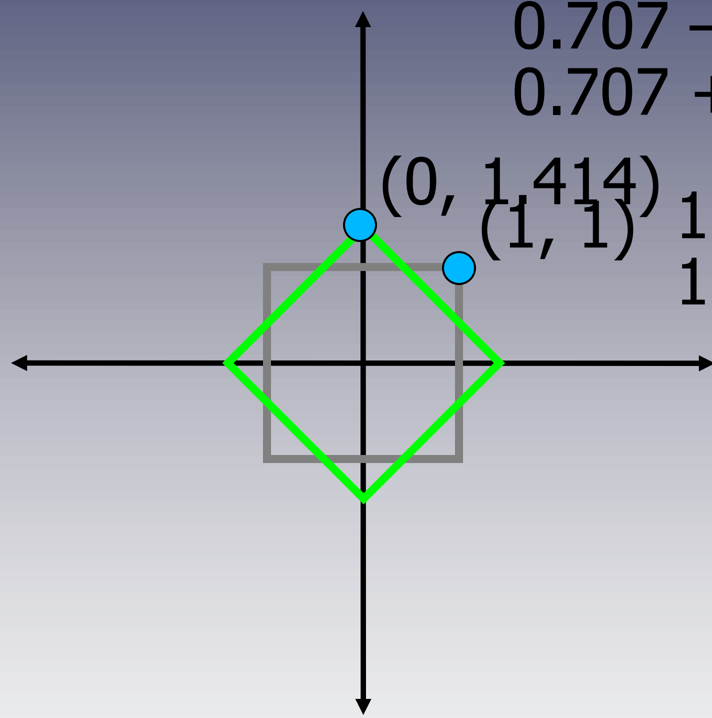
This is a counterclockwise rotation

2-D Rotation



This is a counterclockwise rotation

2-D Rotation: Trigonometry

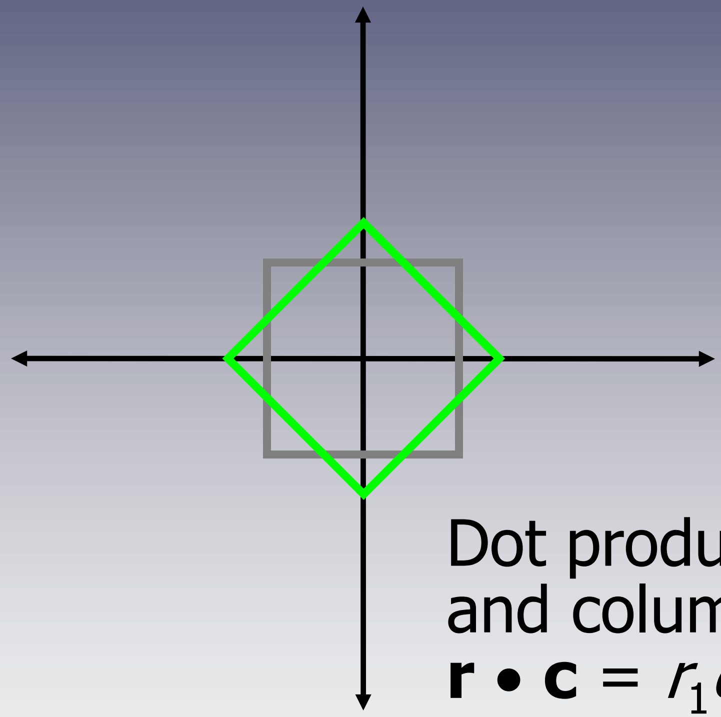


0.707 – 0.707
0.707 + 0.707

1 cos(45) – 1 sin(45)
1 sin(45) + 1 cos(45)

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{pmatrix}$$

2-D Rotation: Matrix Multiplication

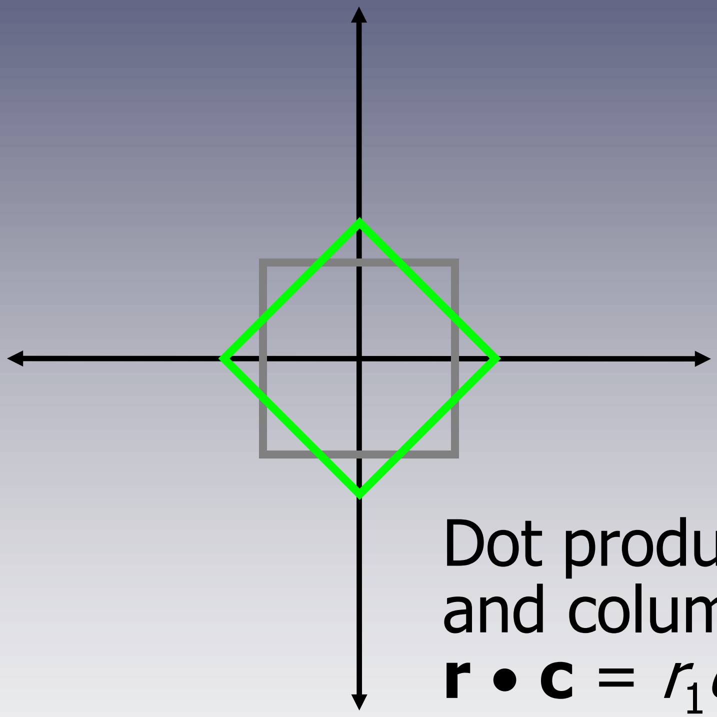


Dot product • of row vector and column vector:

$\mathbf{r} \bullet \mathbf{c} = r_1 c_1 + r_2 c_2 + \ldots + r_n c_n$

$$\begin{pmatrix} \boxed{x'} \\ y' \end{pmatrix} = \begin{pmatrix} \boxed{\cos\theta \quad -\sin\theta} \\ \sin\theta \quad \cos\theta \end{pmatrix} \begin{pmatrix} \boxed{\begin{matrix} x \\ y \end{matrix}} \end{pmatrix}$$

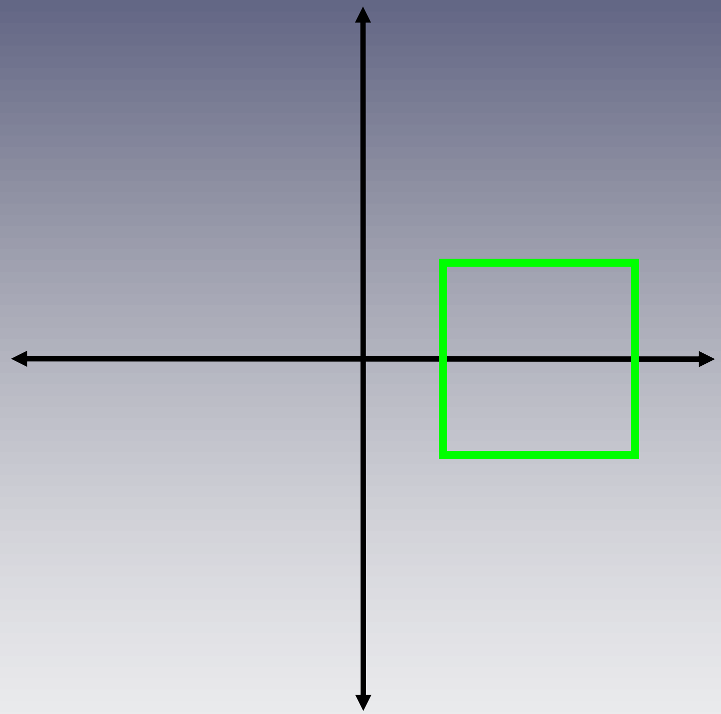
2-D Rotation: Matrix Multiplication



Dot product • of row vector and column vector:
$\mathbf{r} \bullet \mathbf{c} = r_1 c_1 + r_2 c_2 + \ldots + r_n c_n$

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

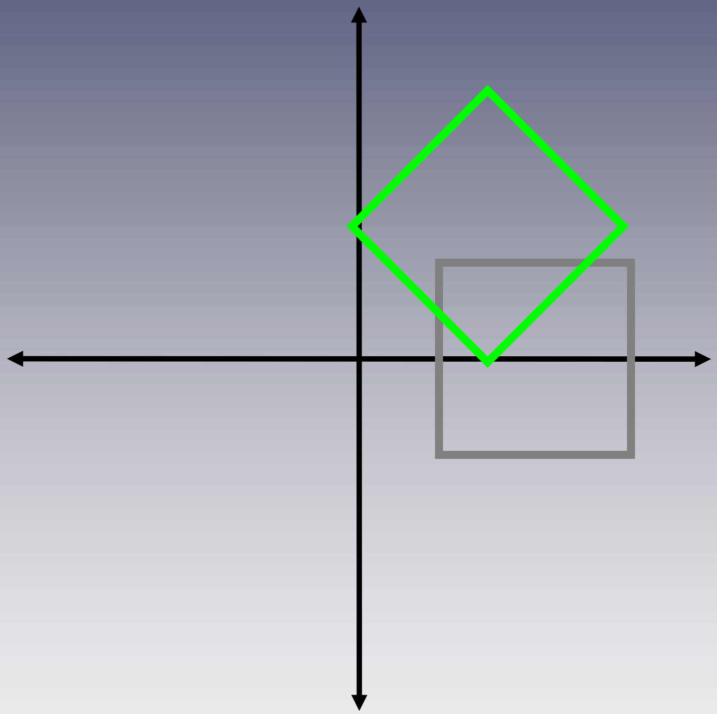
2-D Rotation (uncentered)



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

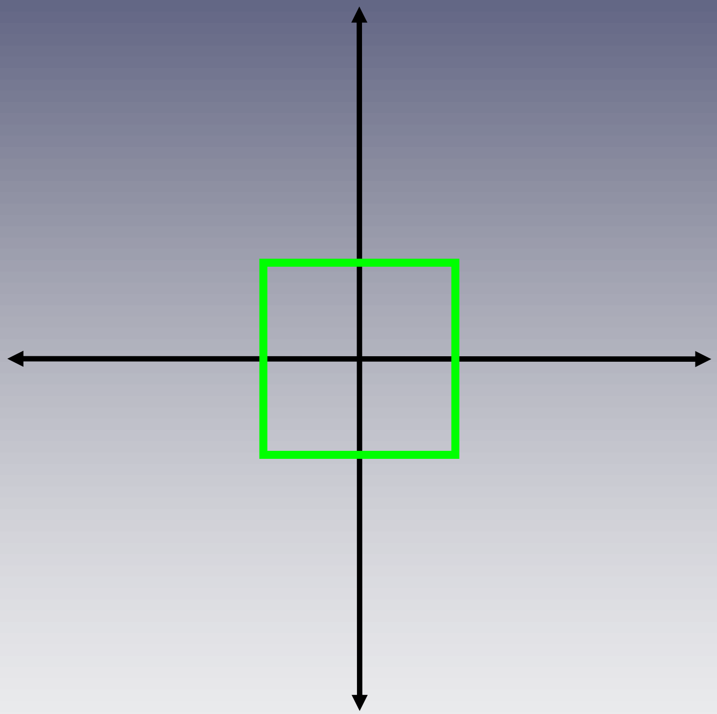
University of Delaware

2-D Rotation (uncentered)

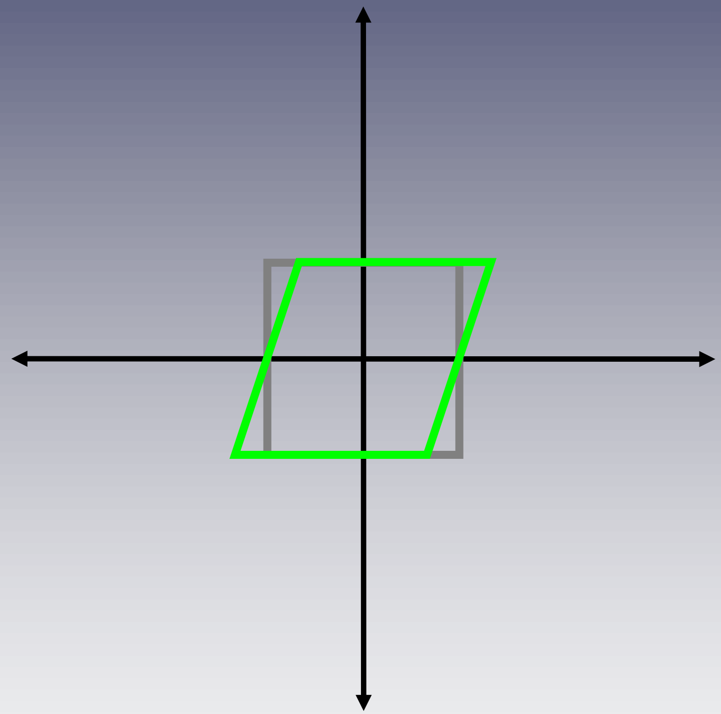


$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

2-D Shear (horizontal)

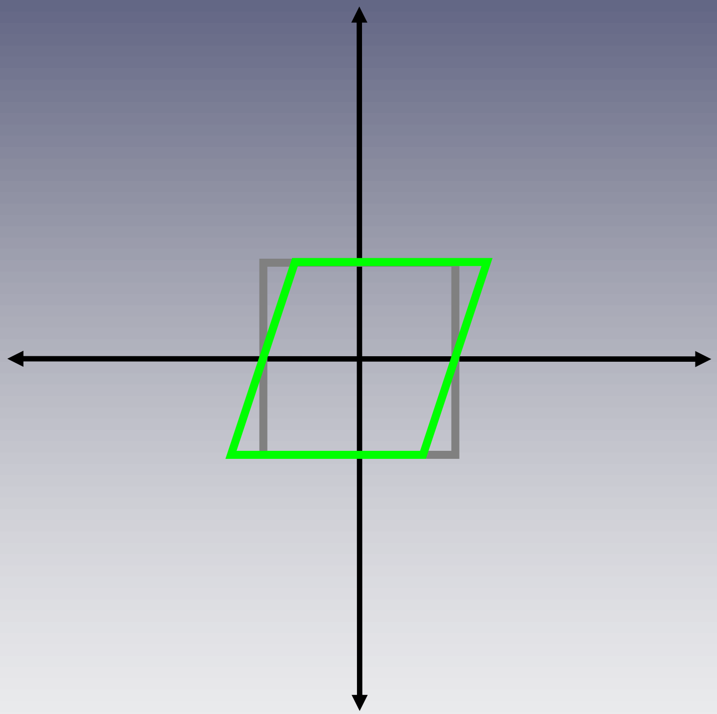


2-D Shear (horizontal)



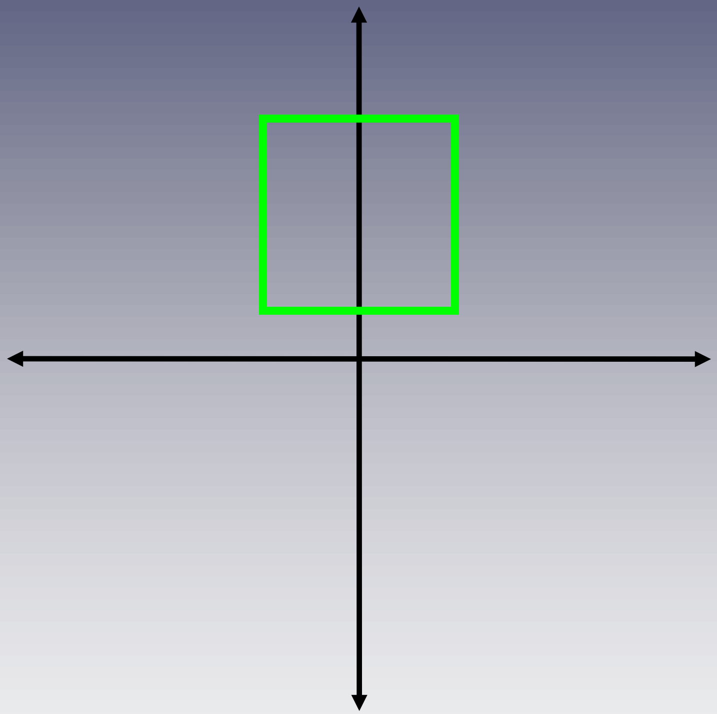
Horizontal displacement proportional to vertical position

2-D Shear (horizontal)

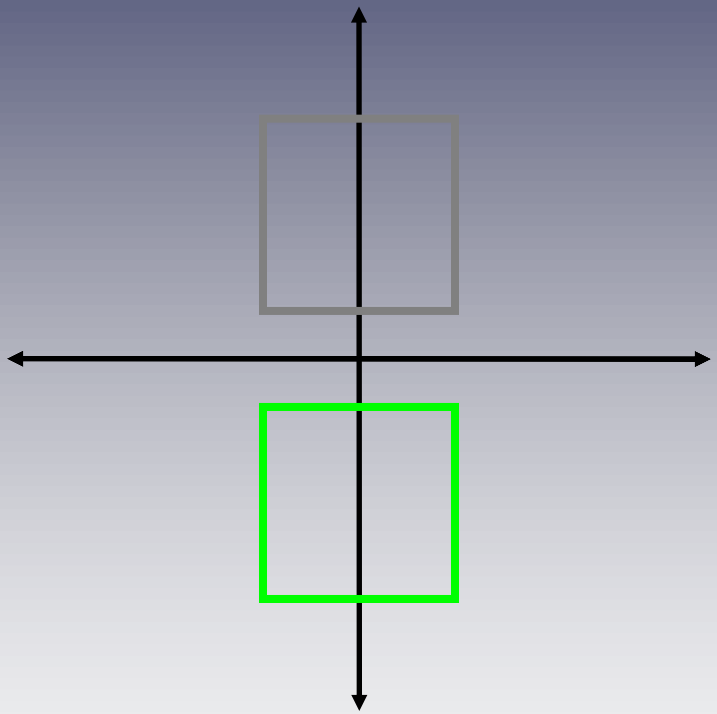


$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

2-D Reflection (vertical)

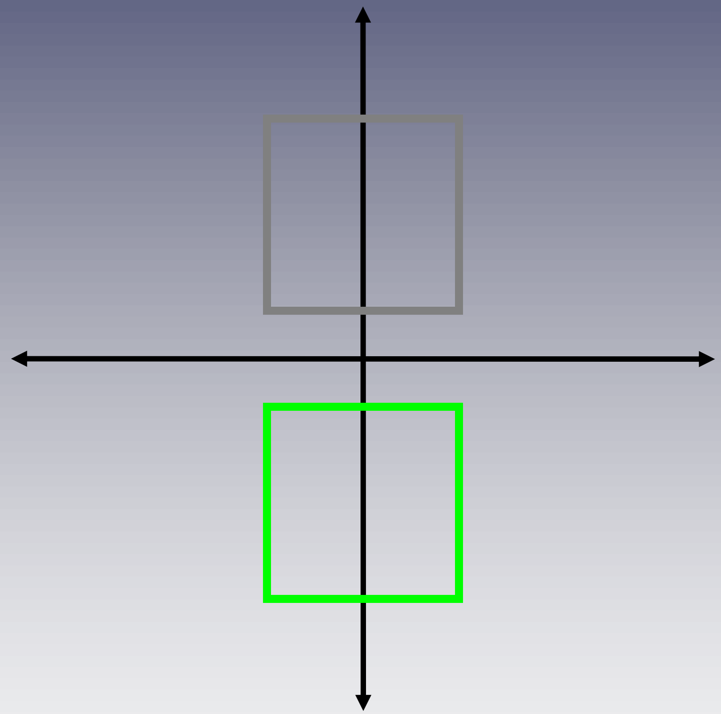


2-D Reflection (vertical)



Just a special case of scaling—”negative” scaling

2-D Reflection (vertical)



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

2-D Transformations: OpenGL

- 2-D transformation functions
  - **`glTranslate(x, y, 0)`**
  - **`glScale(sx, sy, 0)`**
  - **`glRotate(theta, 0, 0, 1)`** (angle in degrees; direction is counterclockwise)
- No explicit shear built into OpenGL

Representing Transformations

- Note that we've defined rotation, scaling, etc. as matrix multiplications, but translation as a vector addition

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

- It's inconvenient (inelegant?) to have two different operations (addition and multiplication) for different forms of transformation
- It would be desirable for all transformations to be expressed in a common form
  - **Solution: Homogeneous coordinates**

Homogeneous Coordinates

- Note: write vectors vertically instead of horizontally
- Let $\mathbf{x} = (x_1, \dots, x_n)^T$ be a point in Euclidean space
- Change to *homogeneous* coordinates:

$$\mathbf{x} \Rightarrow (\mathbf{x}^T, 1)^T$$

- Think of last coordinate w as a **scale** coordinate, with $w = 1$ being the default scale
- Can go back to non-homogeneous representation by normalizing (some transformations may change scale):

$$(\mathbf{x}^T, w)^T \Rightarrow \mathbf{x} / w$$

Example: Translation with homogeneous coordinates

- Old way: $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$
- New way:

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

$$\mathbf{x}' = \begin{pmatrix} \mathbf{Id} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \mathbf{x}$$

Homogeneous Coordinates: Rotations, etc.

- A 2-D rotation, scaling, shear or other transformation normally expressed by a 2 x 2 matrix **R** is written in homogeneous coordinates with the following 3 x 3 matrix:

$$\mathbf{x}' = \begin{pmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix} \mathbf{x}$$

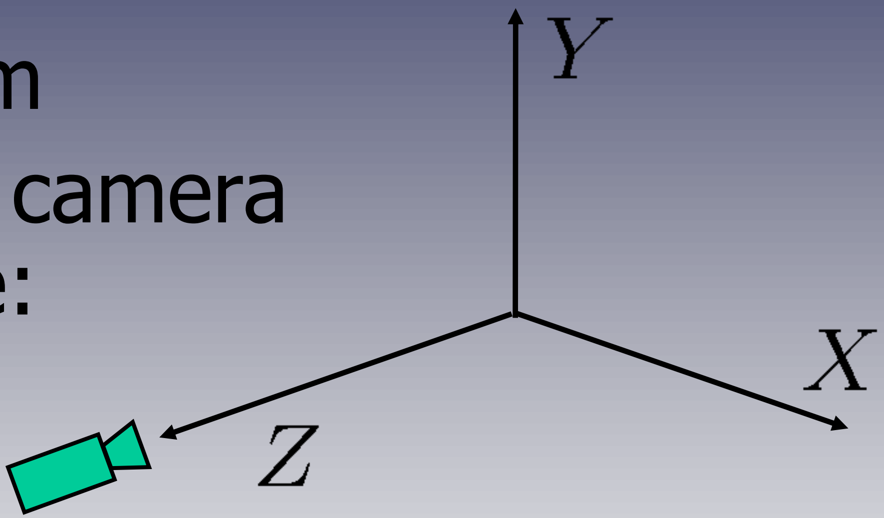
- "Compose" transforms by multiplying their matrices together
- Matrix multiplication's non-commutativity explains why **RTx** ≠ **TRx**

OpenGL's coordinates

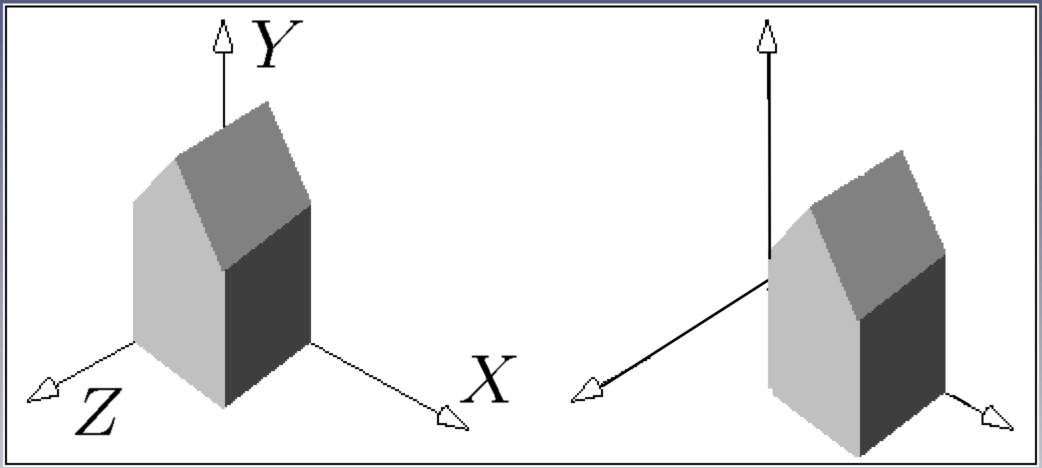
- The underlying form of all points/vertices is a 4-D vector (x, y, z, w)
- If you do something in 2-D, OpenGL simply sets $z = 0$ for you
- If the scale coordinate w is not set explicitly (recall that there is a **`glVertex4()`** that allows you to do so), OpenGL sets $w = 1$ for you

OpenGL 3-D coordinates

- "Right-handed" system
- From point of view of camera looking out into scene:
  - +X right, -X left
  - +Y up, -Y down
  - +Z **behind** camera, -Z in front
  - One way to remember: cross product of +X and +Y axis vectors is +Z
- Positive rotations are counterclockwise around axis of rotation



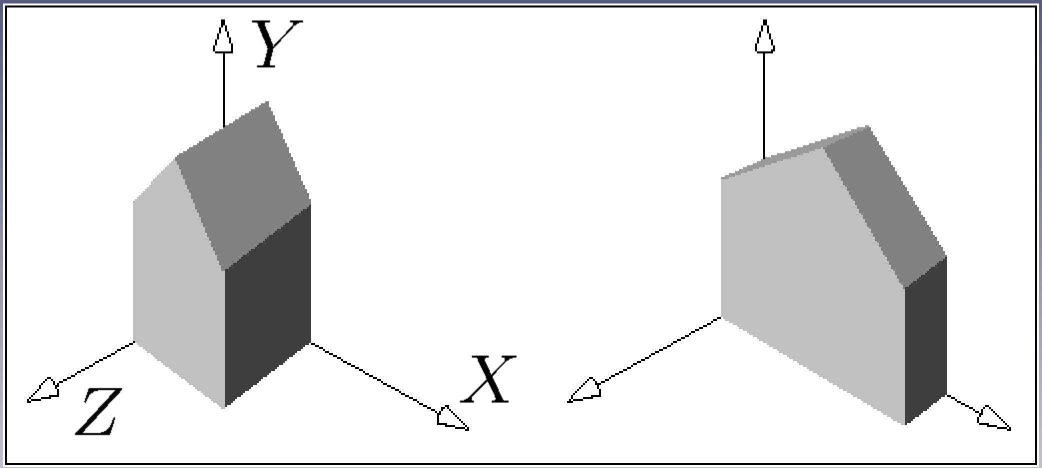
3-D Translations



modified from Hill

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

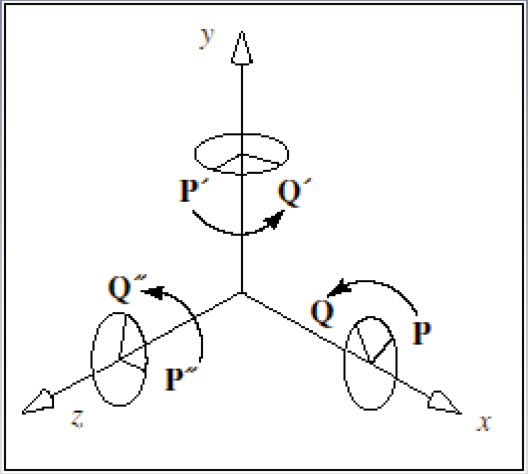
3-D Scaling



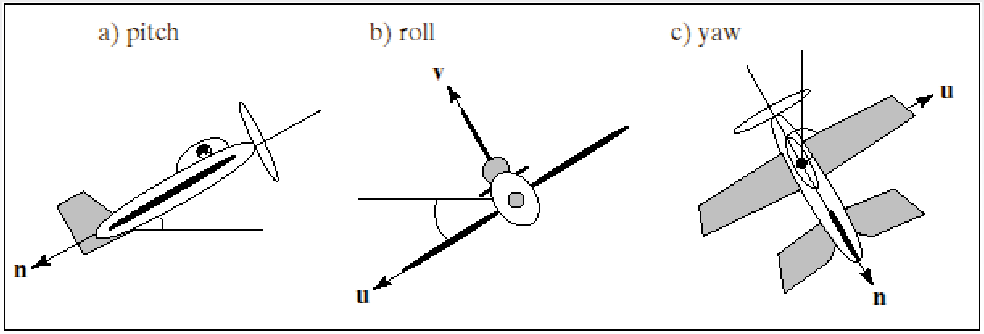
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

3-D Rotations

- In 2-D, we are always rotating in the plane of the image, but in 3-D the axis of rotation itself is a variable
- Three canonical rotation axes are the coordinate axes X, Y, Z
- These are sometimes referred to in aviation terms: **pitch**, **yaw** or **heading**, and **roll**, respectively



from Hill



from Hill

3-D Rotation Matrices

- Similar form to 2-D rotation matrices, but with coordinate corresponding to rotation axis held constant
- E.g., a rotation about the X axis of θ radians:

$$\mathbf{R}_X(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3-D Rotation Matrices

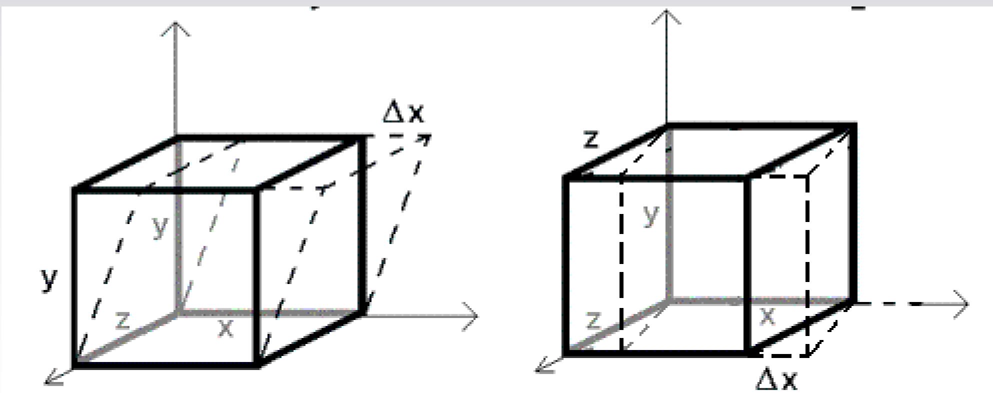
- Similarly:

$$\mathbf{R}_Y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R}_Z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3-D Shears

- Basic form: 4 x 4 identity matrix with one non-zero off-diagonal element in the upper-left 3 x 3 submatrix
- Six possibilities: $\mathbf{H}_{XY}(s)$, $\mathbf{H}_{XZ}(s)$, $\mathbf{H}_{YX}(s)$, $\mathbf{H}_{YZ}(s)$, $\mathbf{H}_{ZX}(s)$, $\mathbf{H}_{ZY}(s)$
  - 1st subscript: Which coordinate is **changed by shear** (row where *s* appears)
  - 2nd subscript: Which coordinate is the **shearing proportional to** (column of *s*)
- E.g.:



$\mathbf{H}_{XY}(s)$ $\mathbf{H}_{XZ}(s)$

$$\mathbf{H}_{XZ}(s) = \begin{pmatrix} 1 & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

OpenGL matrix stacks

- **`GL_MODELVIEW`** matrix is the 3-D transformation matrix we've been talking about
- OpenGL maintains a stack; top matrix is one applied to drawing commands
  - Can "read" with **`glGet(GL_MODELVIEW_MATRIX)`**
- So pushing and popping (the right stack) save and restore transformations
  - **`glPushMatrix()`** pushes a copy of top of stack
- Postmultiplication automatically happens when **`glTranslate()`**, **`glRotate()`**, etc. called successively

OpenGL stacks: Modifying top matrix

- Why?
  - OpenGL has no built-in shear function or many other "exotic" transformations
- Replacement of current matrix
  - **`glLoadIdentity()`**
  - **`glLoadMatrix(M)`**
- Postmultiplication of current matrix
  - **`glMultMatrix(M)`**

glLoadMatrix(), glMultMatrix()

- Allocate and initialize an array of 16 doubles or floats **m**
- Column-major format:

$$\begin{pmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{pmatrix}$$