3D Graphics

- Goal: To produce 2D images of a mathematically described 3D environment
- Issues:
  - Describing the environment: *Modeling* (mostly later)
  - Computing the image: *Rendering*

Graphics Toolkits

- Graphics toolkits typically take care of the details of producing images from geometry
- Input:
  - Where the objects are located and what they look like
  - Where the camera is and how it behaves
  - Parameters for controlling the rendering
- Output: Pixel data in a *framebuffer*
  - Data can be put on the screen
  - Data can be read back for processing (part of toolkit)

OpenGL

- OpenGL is an open standard graphics toolkit
  - Derived from SGI’s GL toolkit
- Provides a range of functions for modelling, rendering and manipulating the framebuffer
- Why use it? Portable, hardware supported, simple and easy to program (really!)
- Alternatives: Direct3D, Java3D - more complex and less well supported respectively

In the Coming Weeks…

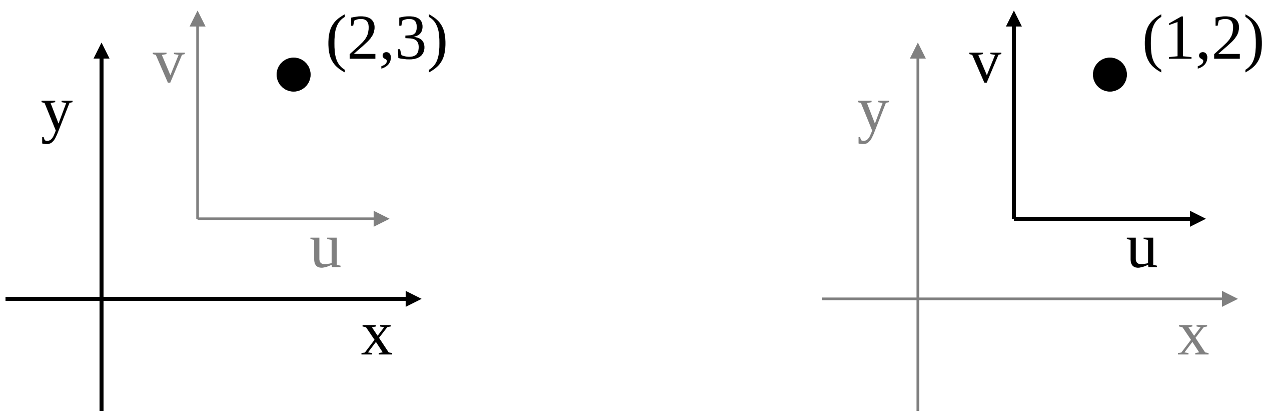
- We will look at the math and algorithms on which OpenGL is built
- We will look at how to access those algorithms in OpenGL

Coordinate Systems

- The use of *coordinate systems* is fundamental to computer graphics
- Coordinate systems are used to describe the locations of points in space
- Multiple coordinate systems make graphics algorithms easier to understand and implement

Coordinate Systems (2)

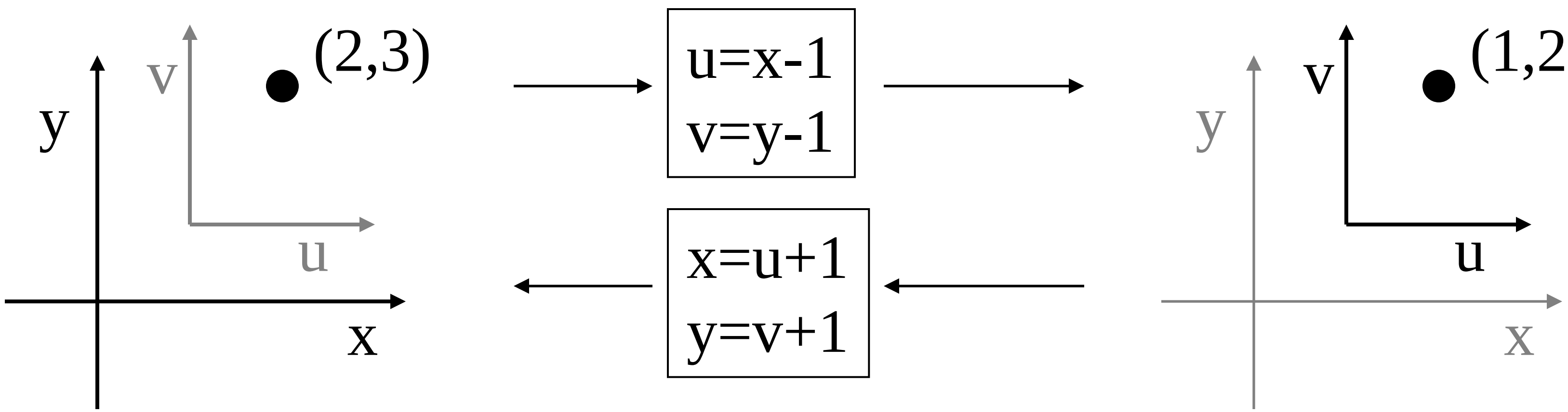
- **Different coordinate systems represent the same point in different ways**



- Some operations are easier in one coordinate system than in another

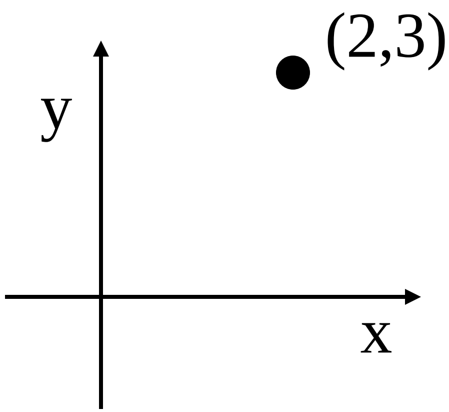
Transformations

- Transformations convert points between coordinate systems



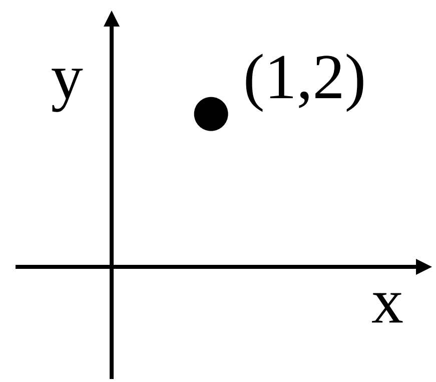
Transformations
(Alternate Interpretation)

- Transformations modify an object's shape and location in one coordinate system



$$x' = x - 1$$
$$y' = y - 1$$

$$x = x' + 1$$
$$y = y' + 1$$



2D Affine Transformations

- An *affine transformation* is one that can be written in the form:

$$x' = a_{xx}x + a_{xy}y + b_x$$
$$y' = a_{yx}x + a_{yy}y + b_y$$

or

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{xx} & a_{xy} \\ a_{yx} & a_{yy} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

Why Affine Transformations?

- Affine transformations are *linear*
  - Transforming all the individual points on a line gives the same set of points as transforming the endpoints and joining them
  - Interpolation is the same in either space: Find the halfway point in one space, and transform it. Will get the same result if the endpoints are transformed and then find the halfway point

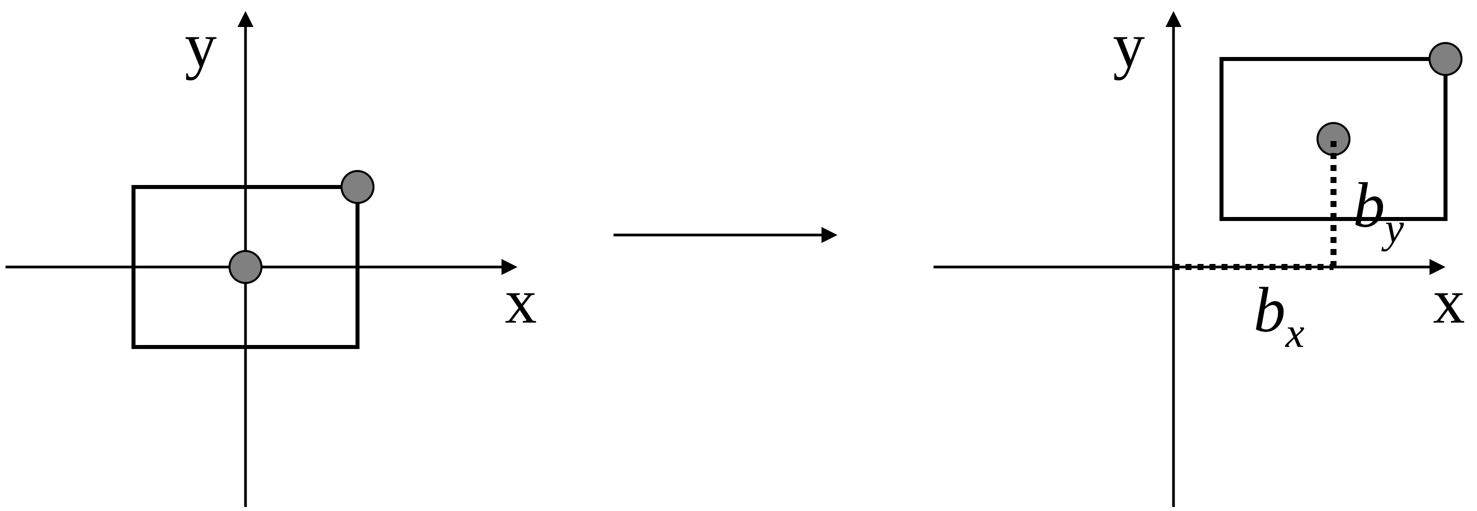
Composition of Affine Transforms

- Any affine transformation can be composed as a sequence of simple transformations:
  - Translation
  - Scaling
  - Rotation
  - Shear
  - Reflection

2D Translation

- Moves an object

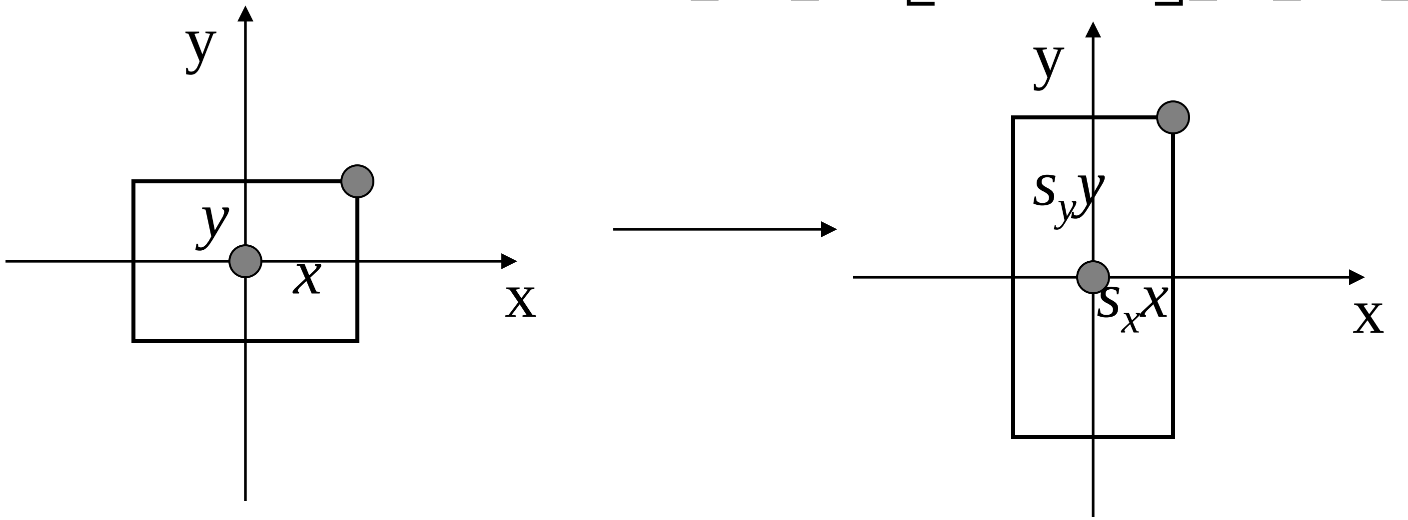
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$



2D Scaling

- Resizes an object in each dimension

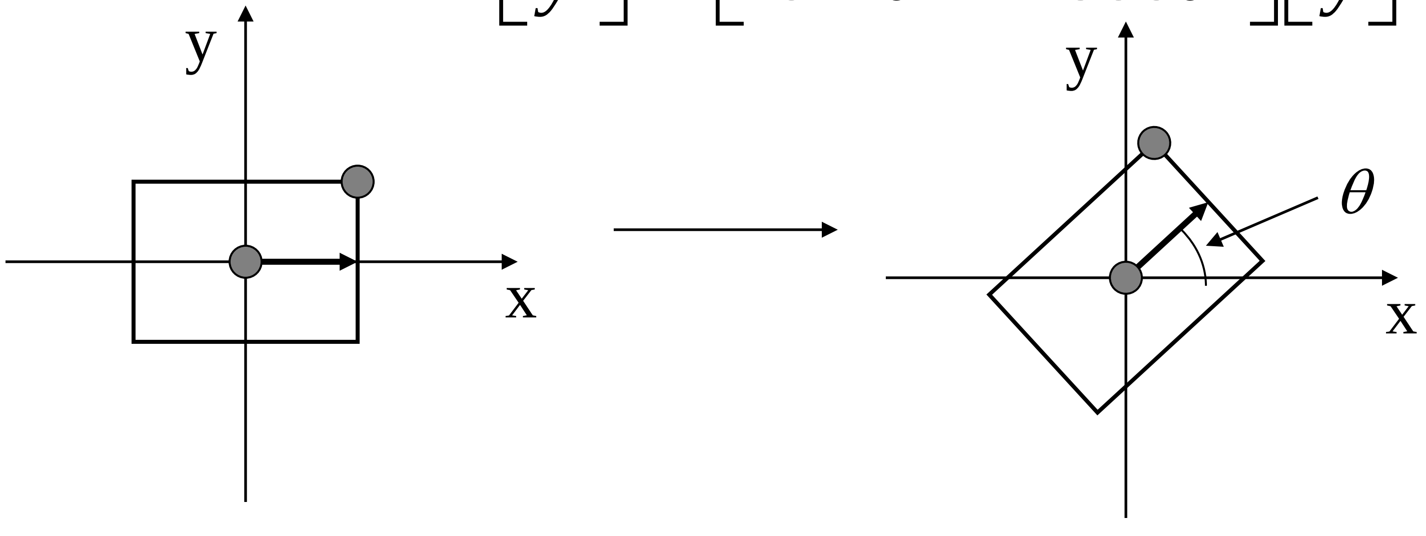
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



2D Rotation

- Rotate counter-clockwise about the origin by an angle θ

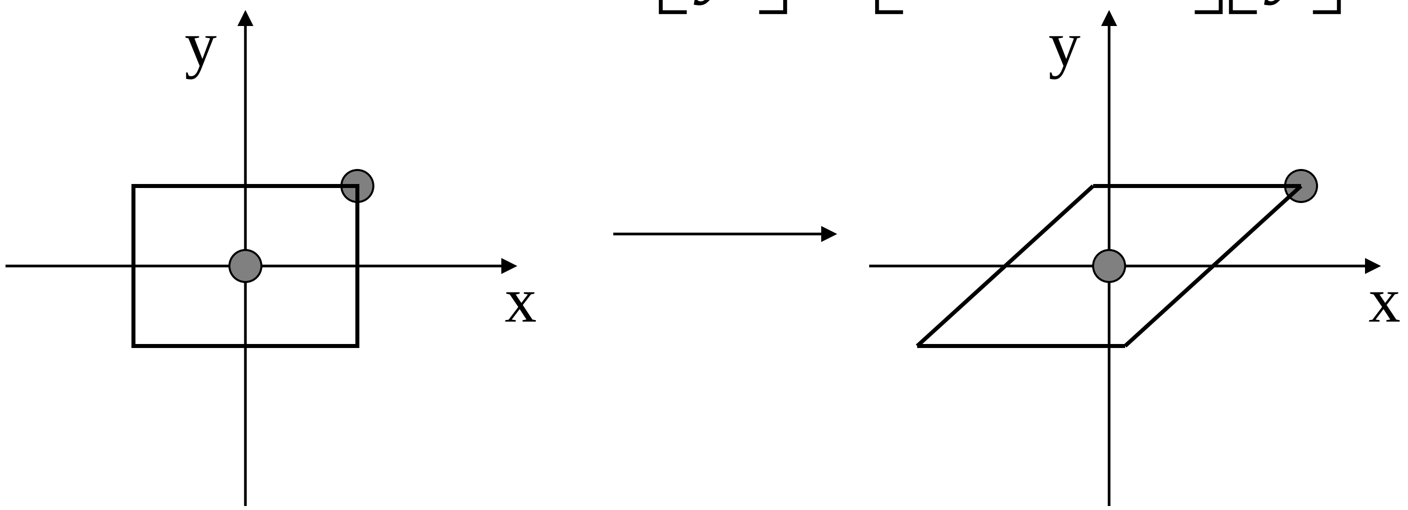
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



X-Axis Shear

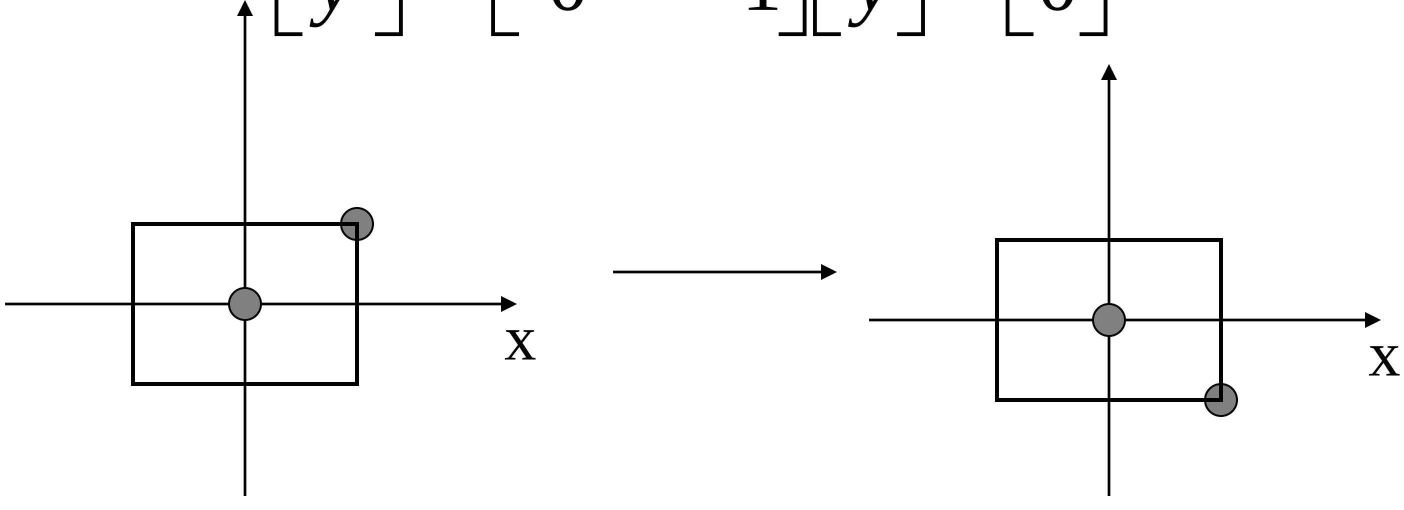
- Shear along x axis (What is the matrix for y axis shear?)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



Reflect About X Axis

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

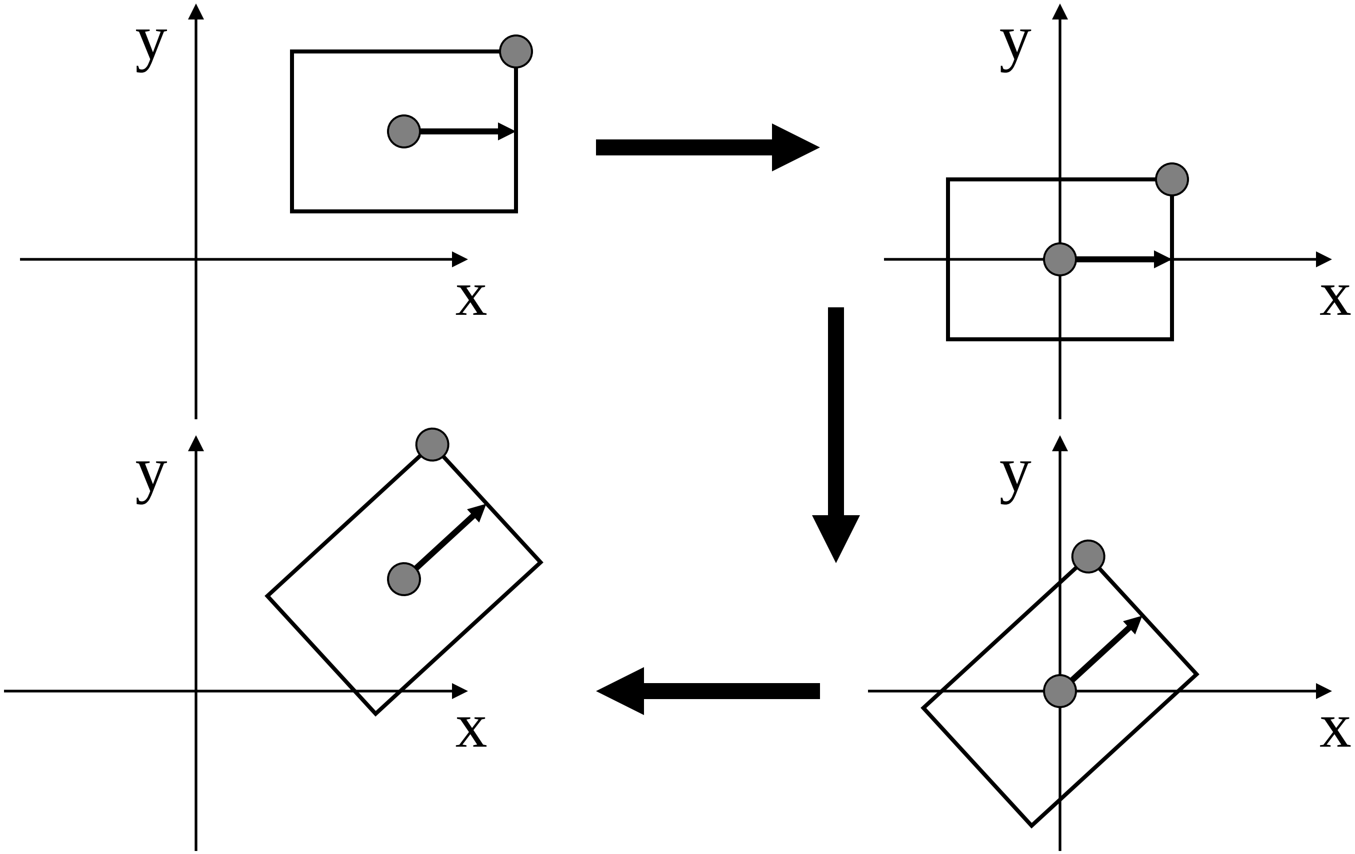


- What is the matrix for reflect about Y axis?

Rotating About An Arbitrary Point

- What happens when you apply a rotation transformation to an object that is not at the origin?
- Solution:
  - Translate the center of rotation to the origin
  - Rotate the object
  - Translate back to the original location

Rotating About An Arbitrary Point



Scaling an Object not at the Origin

- What also happens if you apply the scaling transformation to an object not at the origin?
- Based on the rotating about a point composition, what should you do to resize an object about its own center?

Back to Rotation About a Pt

- Say $\boldsymbol{R}$ is the rotation matrix to apply, and $\boldsymbol{p}$ is the point about which to rotate
- Translation to Origin: $\boldsymbol{x}' = \boldsymbol{x} - \boldsymbol{p}$
- Rotation: $\boldsymbol{x}'' = \boldsymbol{R}\boldsymbol{x}' = \boldsymbol{R}(\boldsymbol{x} - \boldsymbol{p}) = \boldsymbol{R}\boldsymbol{x} - \boldsymbol{R}\boldsymbol{p}$
- Translate back: $\boldsymbol{x}''' = \boldsymbol{x}'' + \boldsymbol{p} = \boldsymbol{R}\boldsymbol{x} - \boldsymbol{R}\boldsymbol{p} + \boldsymbol{p}$
- The translation component of the composite transformation involves the rotation matrix. What a mess!

Homogeneous Coordinates

- Use three numbers to represent a point
- (x,y)=(wx,wy,w) for any constant w≠0
- Typically, (x,y) becomes (x,y,1)
- Translation can now be done with matrix multiplication!

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{xx} & a_{xy} & b_x \\ a_{yx} & a_{yy} & b_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Basic Transformations

- Translation: $\begin{bmatrix} 1 & 0 & b_x \\ 0 & 1 & b_y \\ 0 & 0 & 1 \end{bmatrix}$ Rotation: $\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- Scaling: $\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Homogeneous Transform Advantages

- Unified view of transformation as matrix multiplication
  - Easier in hardware and software
- To compose transformations, simply multiply matrices
  - Order matters: ***AB*** is generally not the same as ***BA***
- Allows for non-affine transformations:
  - Perspective projections!
  - Bends, tapers, many others

3D Transformations Watt Section 1.1

- Homogeneous coordinates: (x,y,z)=(wx,wy,wz,w)
- Transformations are now represented as 4x4 matrices
- Typical graphics packages allow for specification of translation, rotation, scaling and arbitrary matrices
  - OpenGL: glTranslate[fd], glRotate[fd], glScale[fd], glMultMatrix[fd]

3D Translation

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

3D Rotation

- Rotation in 3D is about an *axis* in 3D space passing through the origin
- Using a matrix representation, any matrix with an *orthonormal* top-left 3x3 sub-matrix is a rotation
  - Rows are mutually orthogonal (0 dot product)
  - Determinant is 1
  - Implies columns are also orthogonal, and that the transpose is equal to the inverse

3D Rotation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} & 0 \\ r_{yx} & r_{yy} & r_{yz} & 0 \\ r_{zx} & r_{zy} & r_{zz} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

and for example :

$$r_{xx} r_{yx} + r_{xy} r_{yy} + r_{xz} r_{yz} = 0$$

Problems with Rotation Matrices

- Specifying a rotation really only requires 3 numbers
  - Axis is a unit vector, so requires 2 numbers
  - Angle to rotate is third number
- Rotation matrix has a large amount of redundancy
  - Orthonormal constraints reduce degrees of freedom back down to 3
  - Keeping a matrix orthonormal is difficult when transformations are combined

Alternative Representations

- Specify the axis and the angle (OpenGL method)
  - Hard to compose multiple rotations
- Specify the axis, scaled by the angle
  - Only 3 numbers, but hard to compose
- Euler angles: Specify how much to rotate about X, then how much about Y, then how much about Z
  - Hard to think about, and hard to compose
- Quaternions

Quaternions

- 4-vector related to axis and angle, unit magnitude
  - Rotation about axis (n_x,n_y,n_z) by angle θ:

$$\left(n_x \cos(\theta/2), n_y \cos(\theta/2), n_z \cos(\theta/2), \sin(\theta/2)\right)$$

- Easy to compose
- Easy to go to/from rotation matrix
- See Watt section 17.2.4 and start of 17.2.5

Other Rotation Issues

- Rotation is about an axis at the origin
  - Use the same trick as in 2D: Translate to origin, rotate, and translate back again
- Rotation is not commutative
  - **Rotation order matters**
  - Experiment to convince yourself of this

Transformation Tidbits

- Scale, shear etc extend naturally from 2D to 3D
- Rotation and Translation are the *rigid-body transformations:*
  - Do not change lengths or angles, so a body does not deform when transformed

Thanks