Computer Graphics

Prof. Feng Liu

Fall 2021

http://www.cs.pdx.edu/~fliu/courses/cs447/

10/20/2021

Last time

- Compositing
- NPR

Today

- 3D Graphics Toolkits
  - Transformations
  - 3D Transformations
- Homework 3 available, due 11/01 (Monday)
- Mid-term: in class, 11/03
  - Online exam

Where to now…

- We are now done with 2D images
- We will spend several weeks on the mechanics of 3D graphics
  - Coordinate systems and Viewing
  - Clipping
  - Drawing lines and polygons
  - Lighting and shading

Graphics Toolkits

- Graphics toolkits typically take care of the details of producing images from geometry
- Input (via API functions):
  - Where the objects are located and what they look like
  - Where the camera is and how it behaves
  - Parameters for controlling the rendering
- Functions (via API):
  - Perform well defined operations based on the input environment
- Output: Pixel data in a *framebuffer* – an image in a special part of memory
  - 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 modeling, rendering and manipulating the framebuffer
- Alternatives: Direct3D, Java3D - more complex and less well supported

- What makes a good toolkit?

A Good Toolkit…

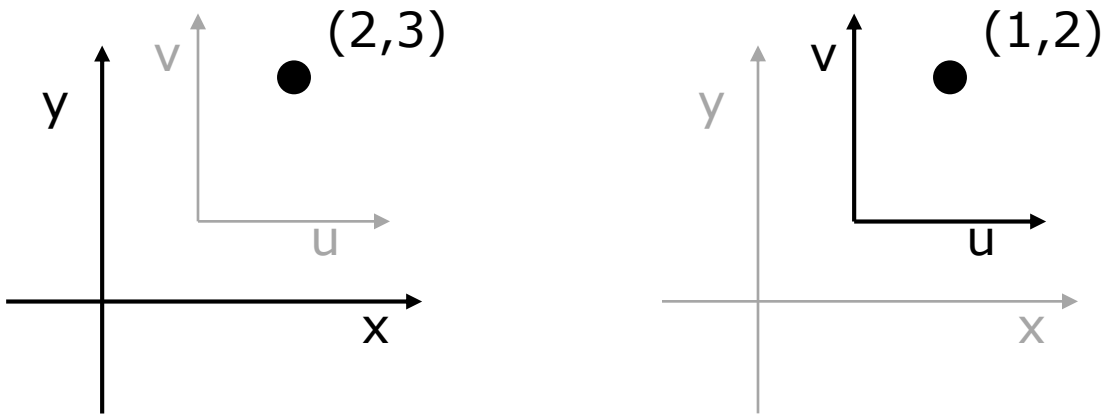
- Everything is a trade-off
- Functionality
  - Compact: a minimal set of commands
  - Orthogonal: commands do different things and can be combined in a consistent way
  - Speed
- Ease-of-Use and Documentation
- Portability
- Extensibility
- Standards and ownership
- Not an exhaustive list …

Coordinate Systems

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

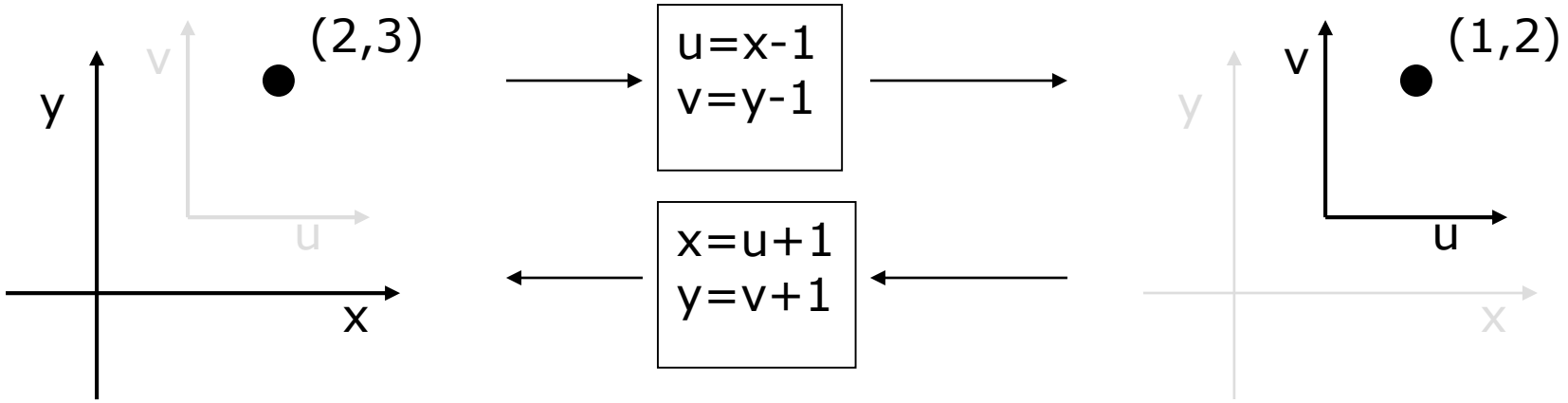
Coordinate Systems (2)

- Different coordinate systems represent the same point in different ways



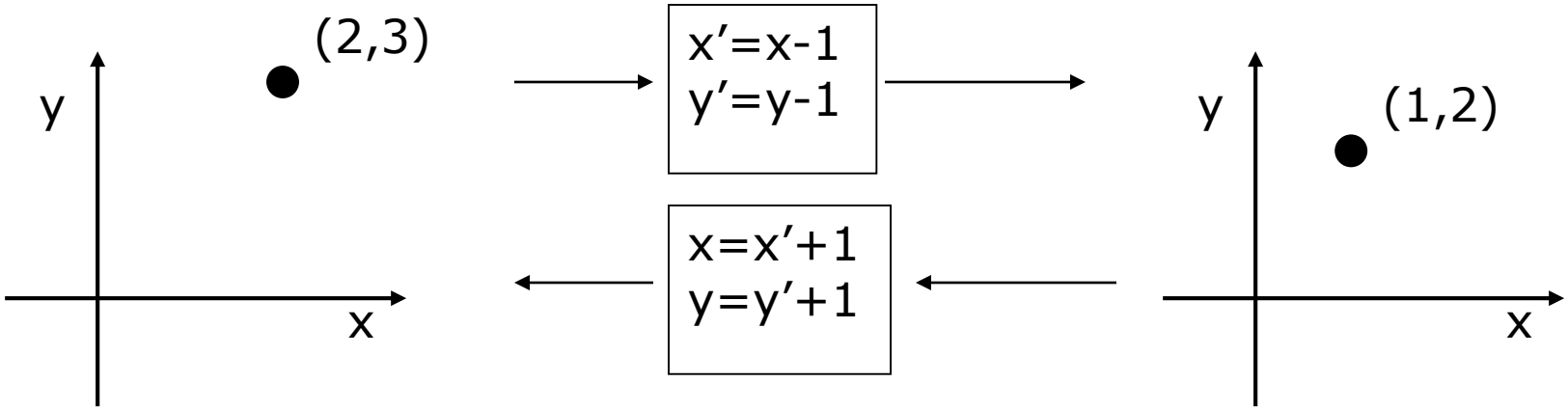
Transformations

- Transformations convert points between coordinate systems



Transformations (Alternate Interpretation)

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

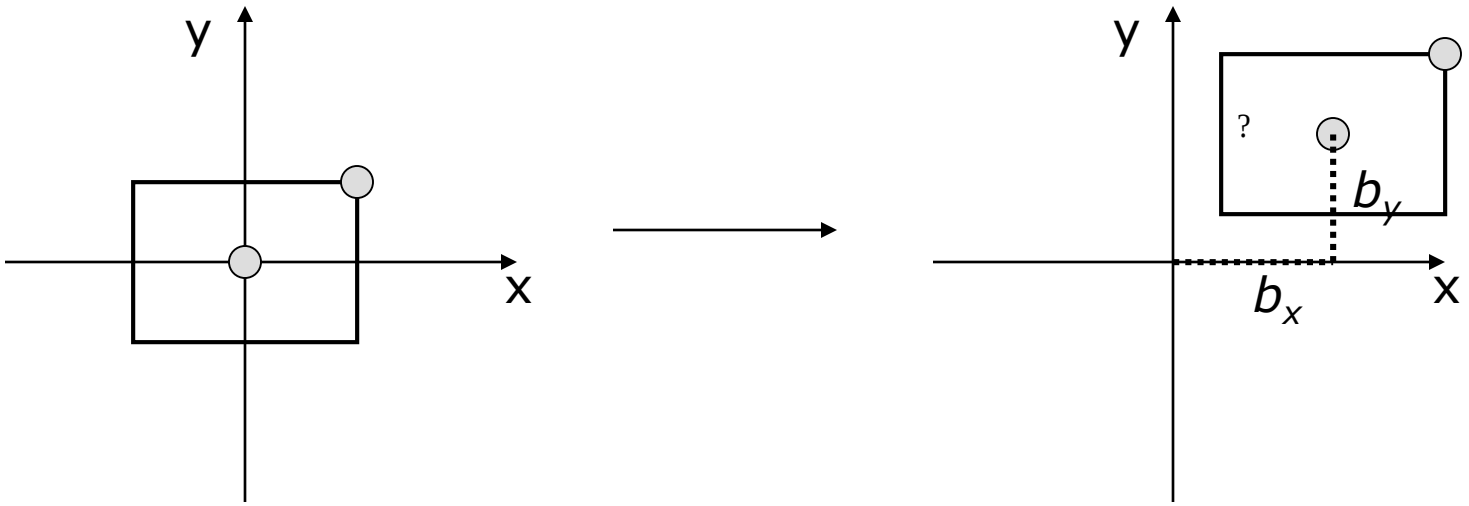


- The previous interpretation is better for some problems, this one is better for others

2D Translation

- Moves an object

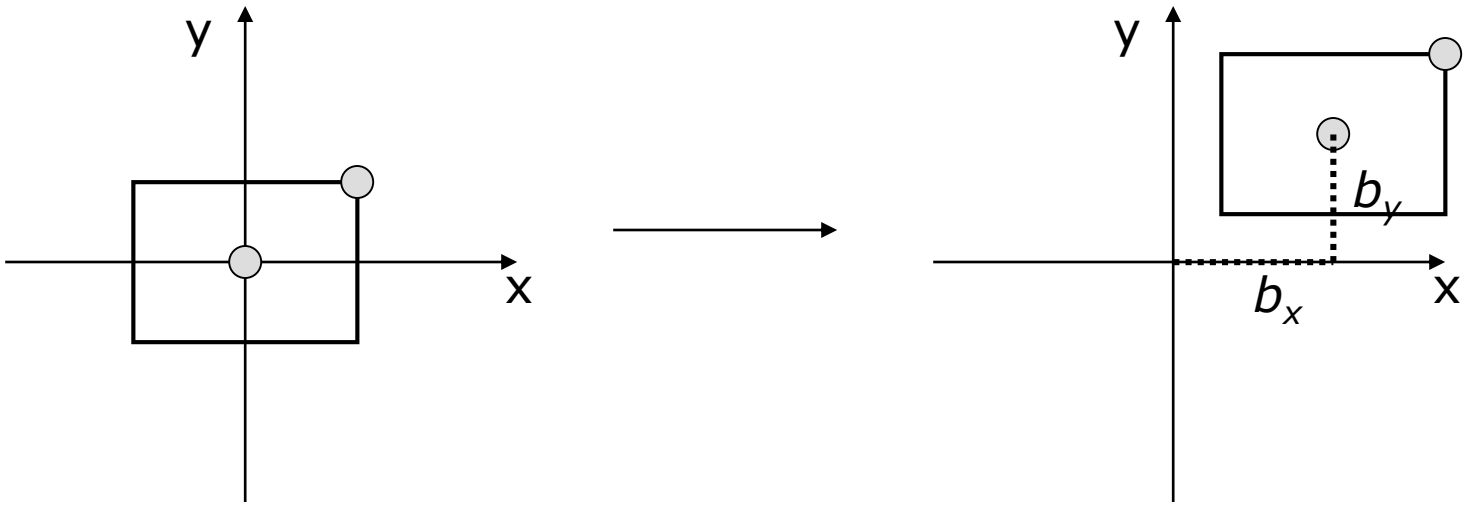
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} ? \\ ? \end{bmatrix}$$



2D Translation

- Moves an object

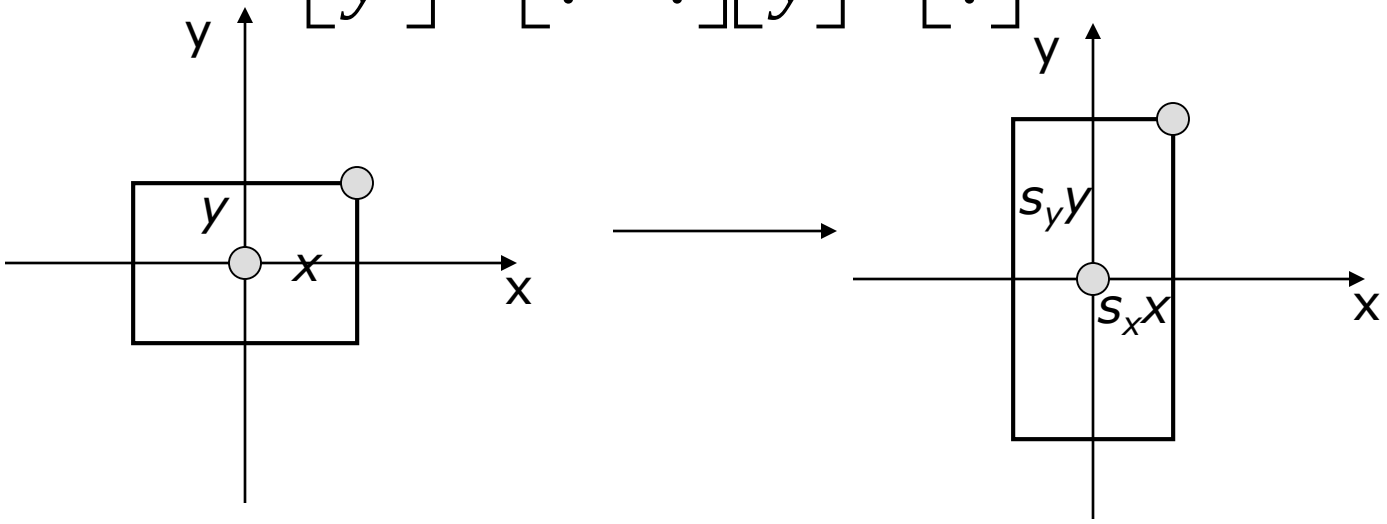
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$



2D Scaling

- Resizes an object in each dimension

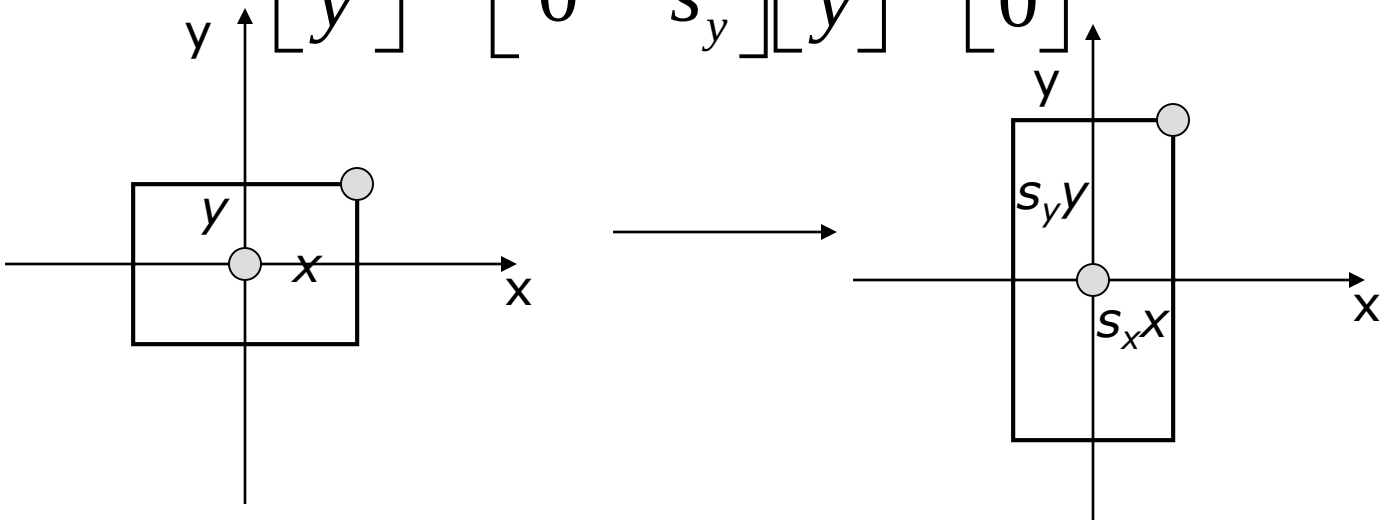
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} ? \\ ? \end{bmatrix}$$



2D Scaling

- Resizes an object in each dimension

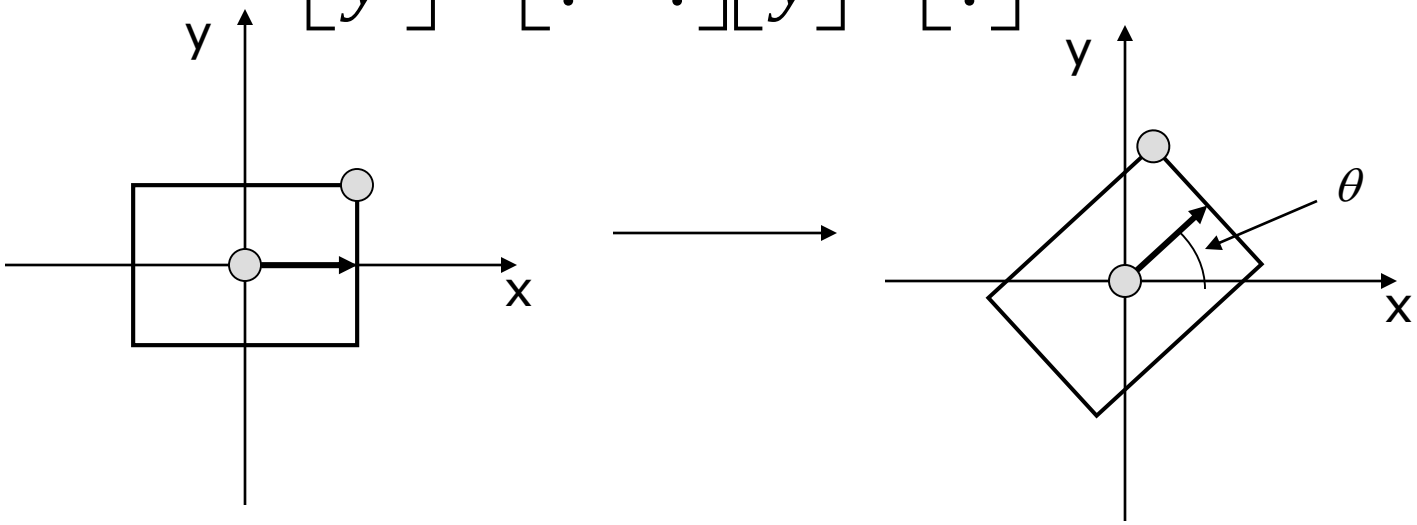
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



2D Rotation

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

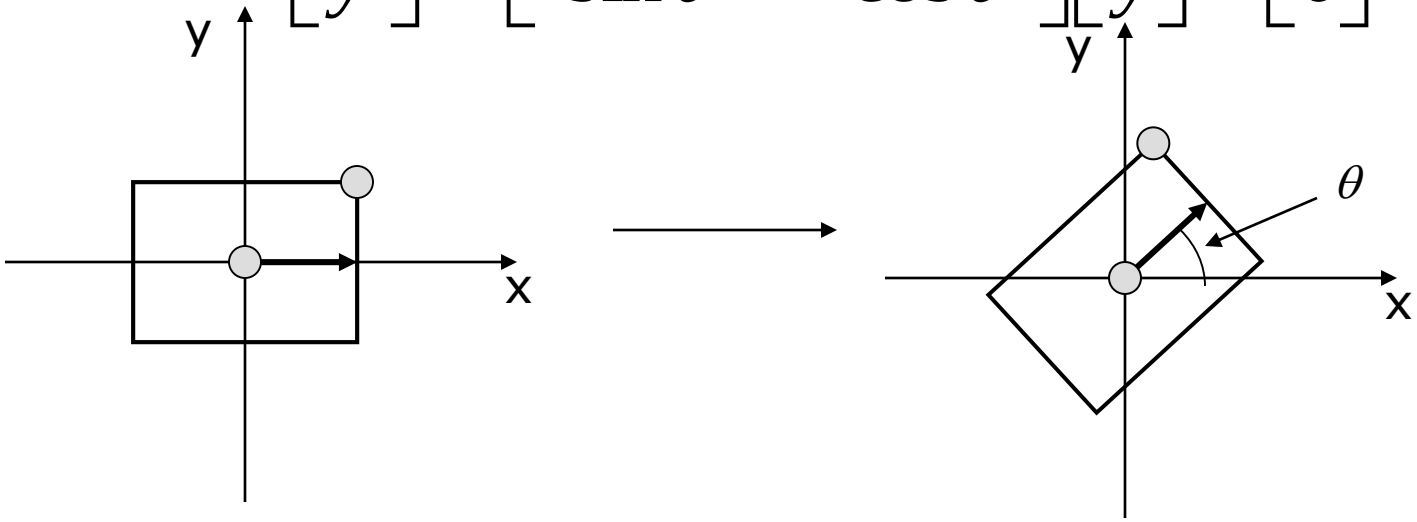
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} ? \\ ? \end{bmatrix}$$



2D Rotation

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

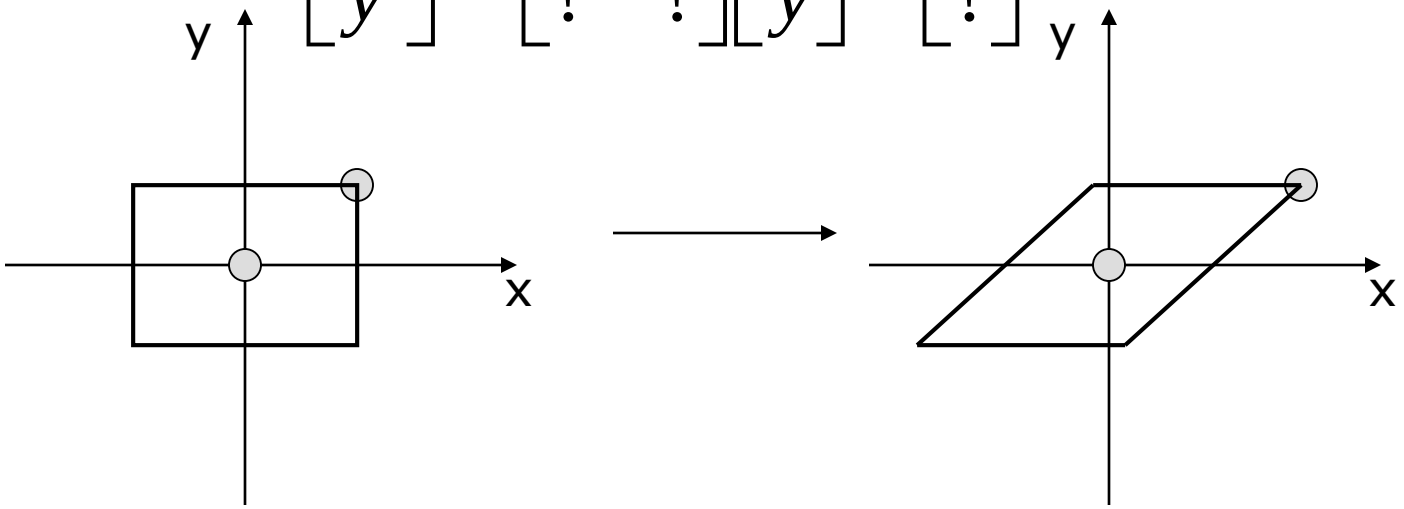
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



X-Axis Shear

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

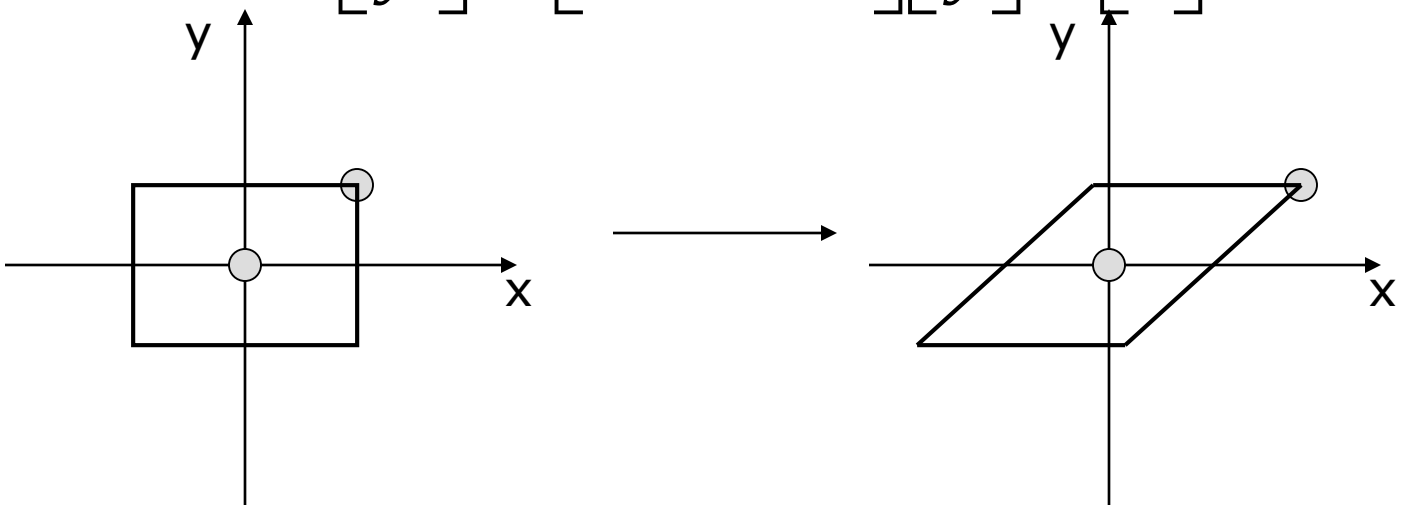
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} ? \\ ? \end{bmatrix}$$



X-Axis Shear

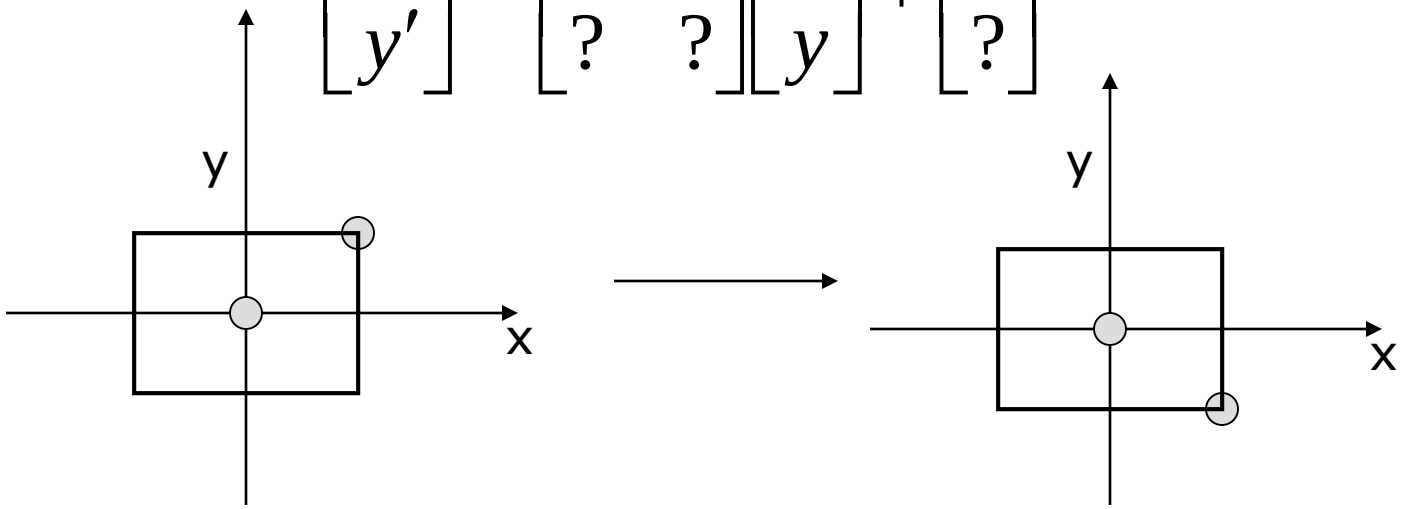
- Shear along x axis (What is the matrix for y axis shear?)

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



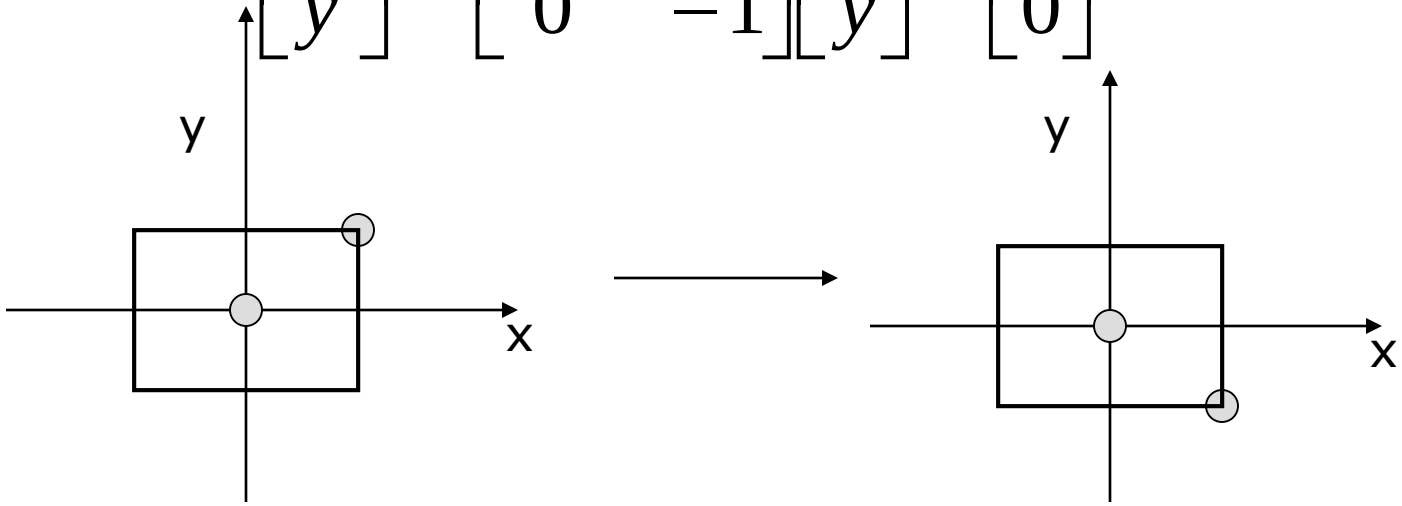
Reflect About X Axis

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} ? \\ ? \end{bmatrix}$$



Reflect About X Axis

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



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

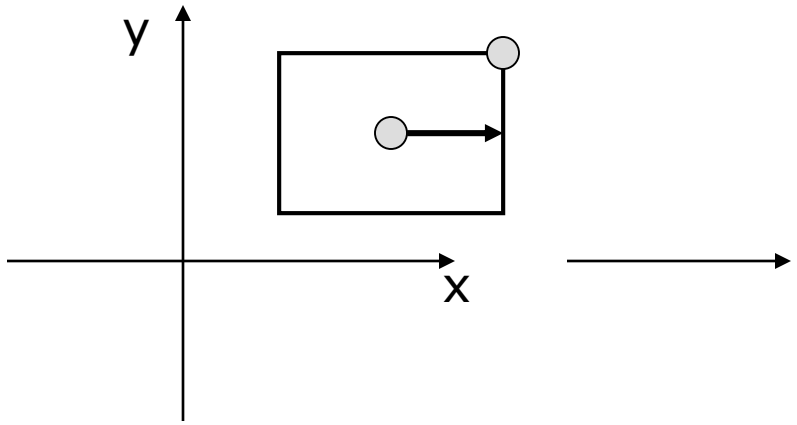
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{xx} & a_{xy} \\ a_{yx} & a_{yy} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

Composition of Affine Transforms

- Any affine transformation can be composed as a sequence of simple transformations:
  - Translation
  - Scaling (possibly with negative values)
  - Rotation

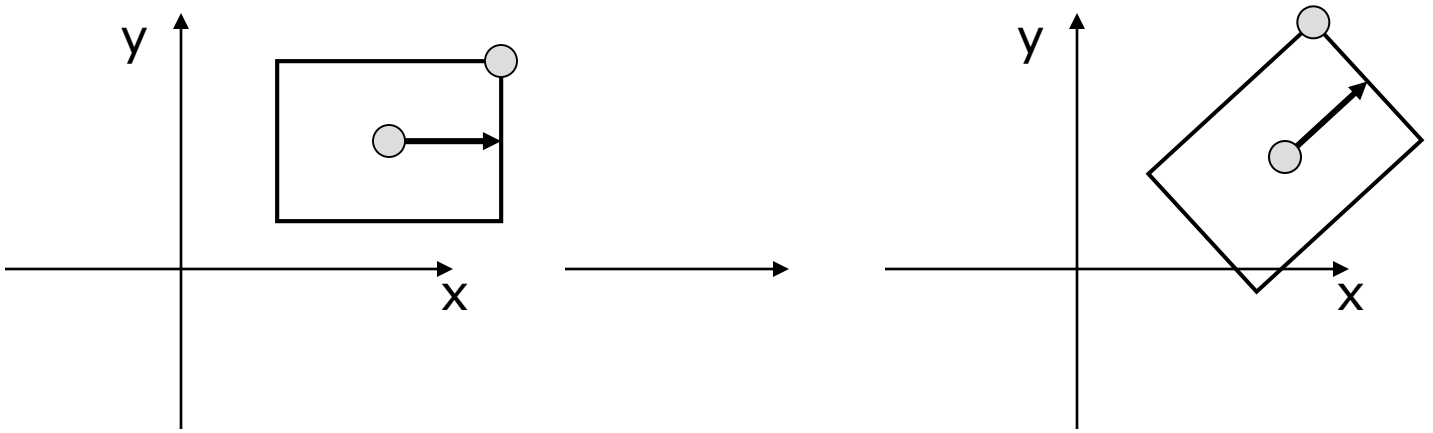
Rotating About An Arbitrary Point

- What happens when you rotate an object about an arbitrary point that is not the origin?



Rotating About An Arbitrary Point

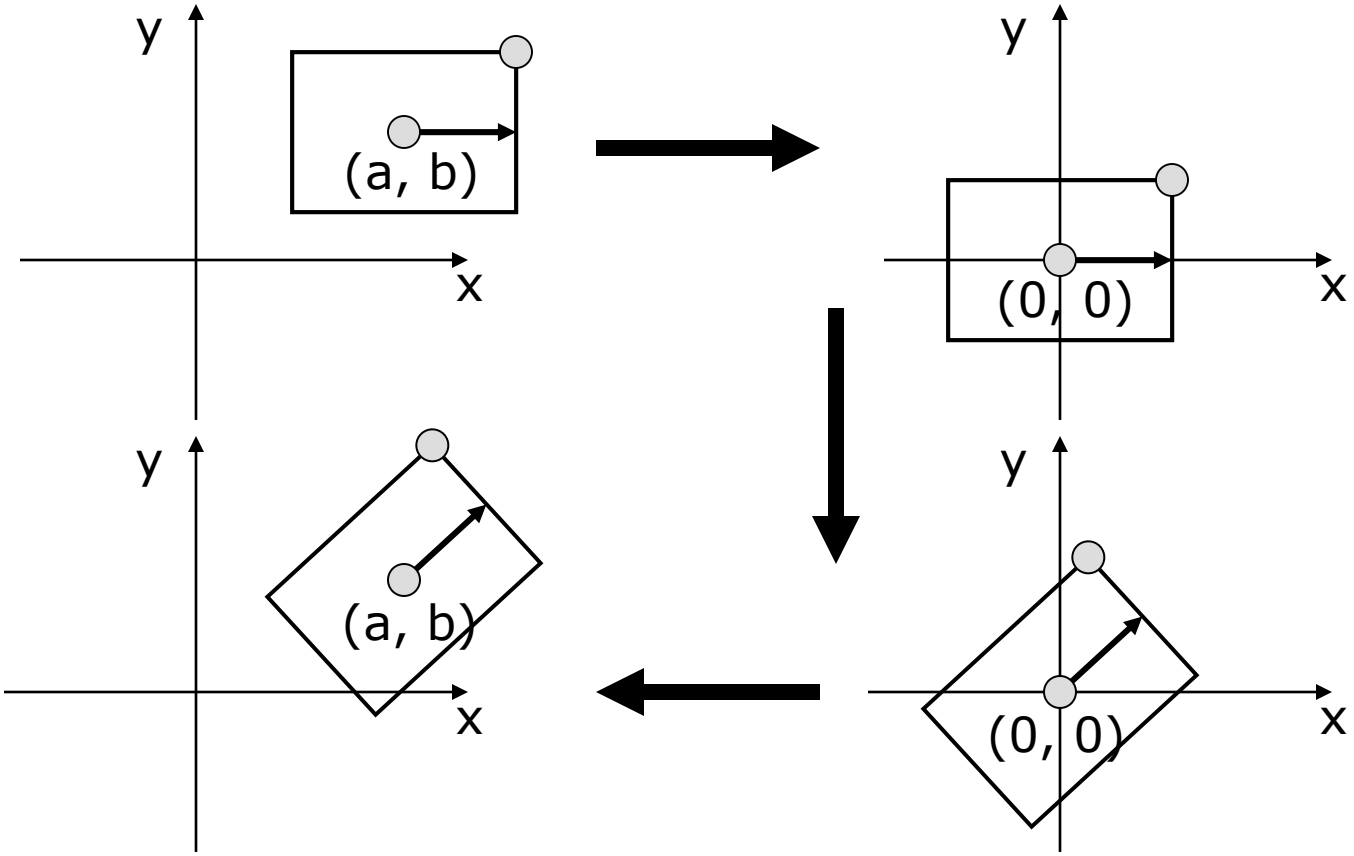
- What happens when you rotate an object about an arbitrary point that is not the origin?



How Do We Compute It?

- How do we rotate an about an arbitrary point?
  - Hint: we know how to rotate about the origin of a coordinate system

Rotating About An Arbitrary Point



Rotating About An Arbitrary Point

- Say you wish to rotate about the point *(a,b)*
- You know how to rotate about *(0,0)*
- Translate so that *(a,b)* is at *(0,0)*
  - $x'=x-a,\ y'=y-b$
- Rotate
  - $x''=(x-a)\cos\theta-(y-b)\sin\theta,\ y''=(x-a)\sin\theta+(y-b)\cos\theta$
- Translate back again
  - $x_f=x''+a,\ y_f=y''+b$

Rotating About An Arbitrary Point

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

Homogeneous Coordinates

- Use three numbers to represent a 2D point
- $(x,y)=(wx,wy,w)$ for any constant $w\neq 0$
  - Typically, (x,y) becomes $(x,y,1)$
  - To go backwards, divide by w
- Translation can now be done with matrix multiplication!

Basic Transformations

Translation: $\begin{bmatrix} 1 & 0 & b_x \\ 0 & 1 & b_y \\ 0 & 0 & 1 \end{bmatrix}$

Rotation: $\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

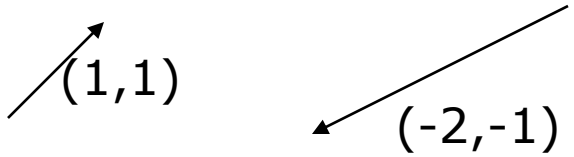
Scaling: $\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

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!

Directions vs. Points

- We have been talking about transforming points
- Directions are also important in graphics
  - Viewing directions
  - Normal vectors
  - Ray directions



- Directions are represented by vectors, like points, and can be transformed, but not like points

Transforming Directions

- Say I define a direction as the difference of two points: $d=a-b$
  - This represents the *direction* of the line between two points
- Now I translate the points by the same amount: $a'=a+t$, $b'=b+t$
- How should I transform d?

Homogeneous Directions

- Translation does not affect directions!
- Homogeneous coordinates give us a very clean way of handling this
- The direction *(x,y)* becomes the homogeneous direction *(x,y,0)*

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

- The correct thing happens for rotation and scaling also
  - Uniform scaling changes the length of the vector, but not the direction

Today

- 3D Graphics Toolkits
  - Transformations
  - 3D Transformations

3D Transformations

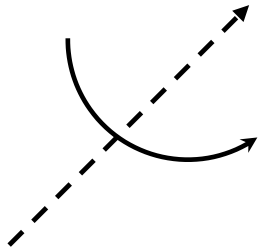
- Homogeneous coordinates: *(x,y,z)=(wx,wy,wz,w)*
- Transformations are now represented as 4x4 matrices
- Typical graphics packages allow for specification of translation, rotation, scaling and arbitrary matrices
  - OpenGL: glTranslate[fd], glRotate[fd], glScale[fd], glMultMatrix[fd]

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 if

$$R = \begin{bmatrix} - & \mathbf{r}_1 & - & 0 \\ - & \mathbf{r}_2 & - & 0 \\ - & \mathbf{r}_3 & - & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \text{ then } \mathbf{r}_1 \bullet \mathbf{r}_2 = 0, \mathbf{r}_1 \bullet \mathbf{r}_3 = 0, \mathbf{r}_2 \bullet \mathbf{r}_3 = 0, \mathbf{r}_1 \bullet \mathbf{r}_1 = 1, etc.$$

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
- Rotations are a very complex subject, and a detailed discussion is way beyond the scope of this course

Alternative Representations

- Specify the axis and the angle (OpenGL method)
- *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
  - Any three axes will do e.g. X,Y,Z
- Specify the axis, scaled by the angle
  - Only 3 numbers, called the *exponential map*
- *Quaternions*

Quaternions

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

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

- Reasonably easy to compose
- Reasonably easy to go to/from rotation matrix
- Only normalized quaternions represent rotations, but you can normalize them just like vectors, so it isn't a problem
- Easy to perform spherical interpolation

Other Rotation Issues

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

Transformation Leftovers

- 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

Modeling 101

- For the moment assume that all geometry consists of points, lines and faces
- Line: A segment between two endpoints

- Face: A planar area bounded by line segments
  - Any face can be *triangulated* (broken into triangles)

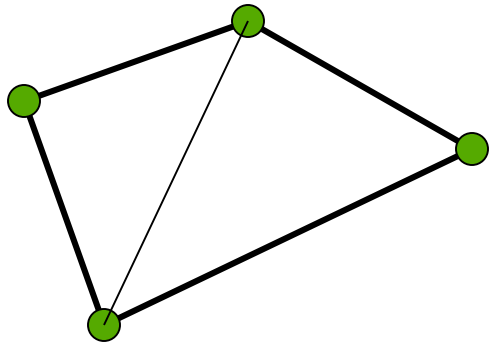


Image url: http://www.capcom-central.com/concept/re4/LeonWireFrame.jpg

Modeling and OpenGL

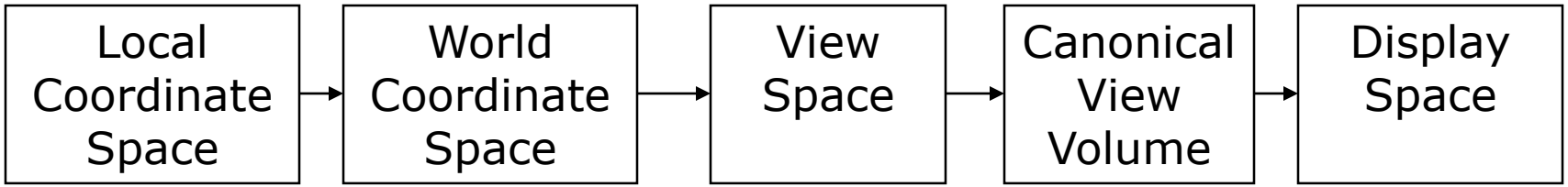
- In OpenGL, all geometry is specified by stating which type of object and then giving the vertices that define it
- `glBegin() …glEnd()`
- `glVertex[34][fdv]`
  - Three or four components (regular or homogeneous)
  - Float, double or vector (eg float[3])
- Chapter 2 of the OpenGL red book

Rendering

- Generate an image showing the contents of some region of space
  - The region is called the *view volume,* and it is defined by the user
- Determine where each object should go in the image
  - *Viewing, Projection*
- Determine which pixels should be filled
  - *Rasterization*
- Determine which object is in front at each pixel
  - *Hidden surface elimination, Hidden surface removal, Visibility*
- Determine what color it is
  - *Lighting, Shading*

Graphics Pipeline

- Graphics hardware employs a sequence of coordinate systems
  - The location of the geometry is expressed in each coordinate system in turn, and modified along the way
  - The movement of geometry through these spaces is considered a pipeline



Next Time

- The Viewing Pipeline
- Perspective Projection
- Clipping
-