CS488

Geometric Transformations

Luc RENAMBOT

Previous Lectures

- Frame buffers
- Drawing a line (Midpoint Line Algorithm)
- Polygon Filling (Edge-table algorithm)
- Line Clipping (Cohen-Sutherland algorithm)
- Polygon Clipping
- Circles
- Geometric Transformations

Now

- At this point we have discussed the primitive operations to set the contents of the frame buffer. Now we are going to go up a level of abstraction and look at how geometric transformations are used to alter the view of a 2D model: how we can translate, scale, and rotate the model, and how transformations affect what the viewport 'sees'
  $\rightarrow$ Geometric Transformations
- Section 5.1 in the textbook: matrices and vectors operations

Geometric Transformations

- How transforms using matrices are used to affect
  - Position
  - Size
  - Orientation of polygons in the scene
- Transforms are applied to vertices and then the edges are drawn between the new vertices

Coordinate Systems

- Right Hand Coordinate System (RHS)
- Left Hand Coordinate System (LHS)
- Point thumb, index finger, and middle finger in orthogonal directions
  - Thumb = x-axis
  - Index = y-axis
  - Middle = z-axis

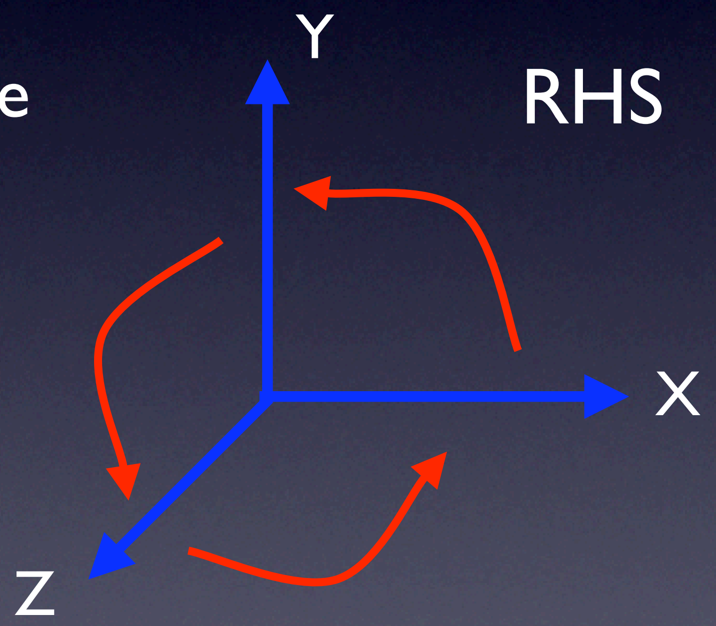
RHS

- Right Hand Coordinate System (RHS)
- Z is coming out of the screen
- Counterclockwise rotations are positive
- If we rotate about the X axis

  the rotation Y→Z is positive
- If we rotate about the Y axis

  the rotation Z→X is positive
- If we rotate about the Z axis

  the rotation X→Y is positive



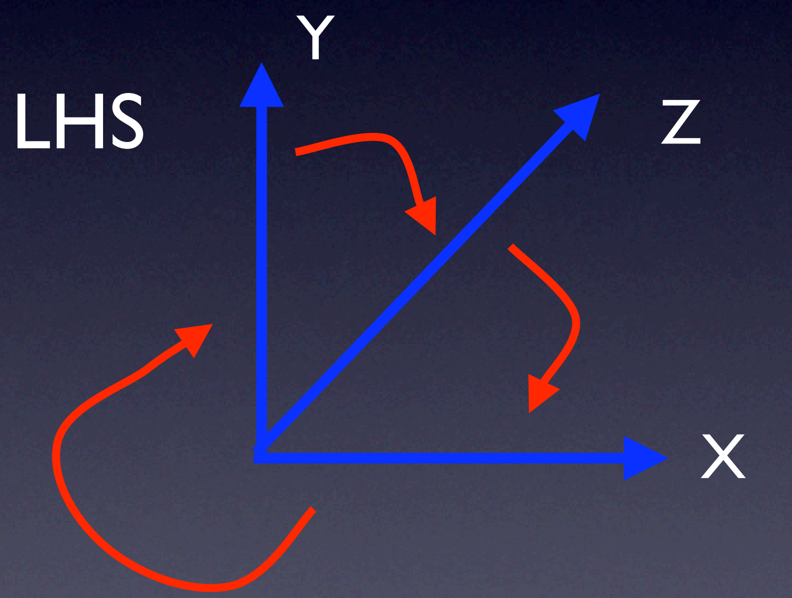
LHS

- Left Hand Coordinate System (LHS)
- Z is going into the screen
- Clockwise rotations are positive
- If we rotate about the X axis

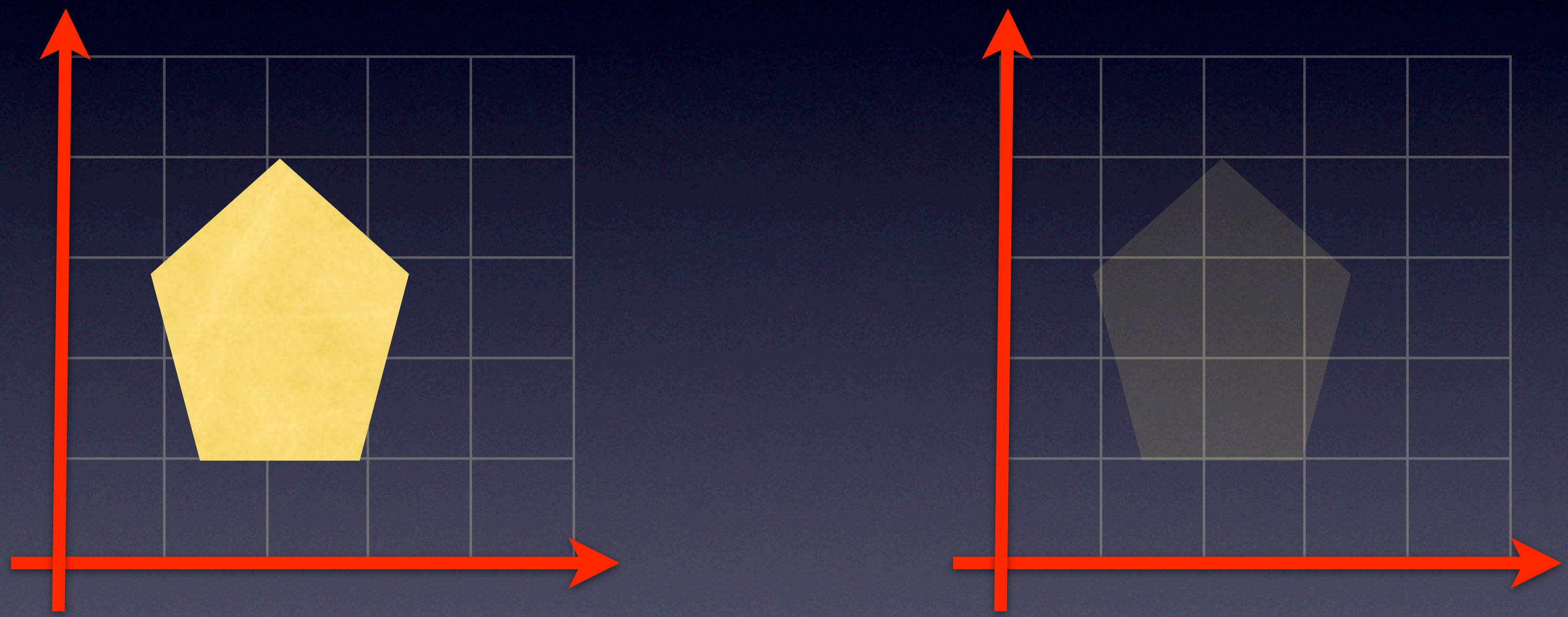
  the rotation Y→Z is positive
- If we rotate about the Y axis

  the rotation Z→X is positive
- If we rotate about the Z axis

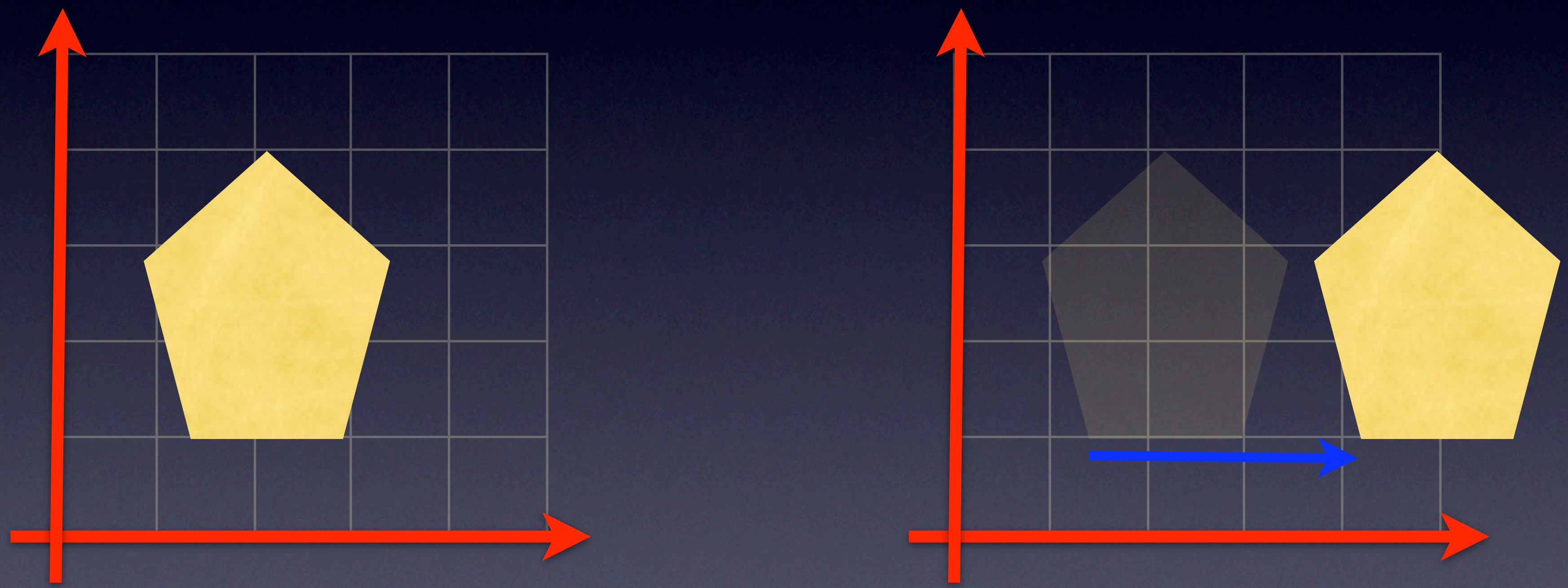
  the rotation X→Y is positive



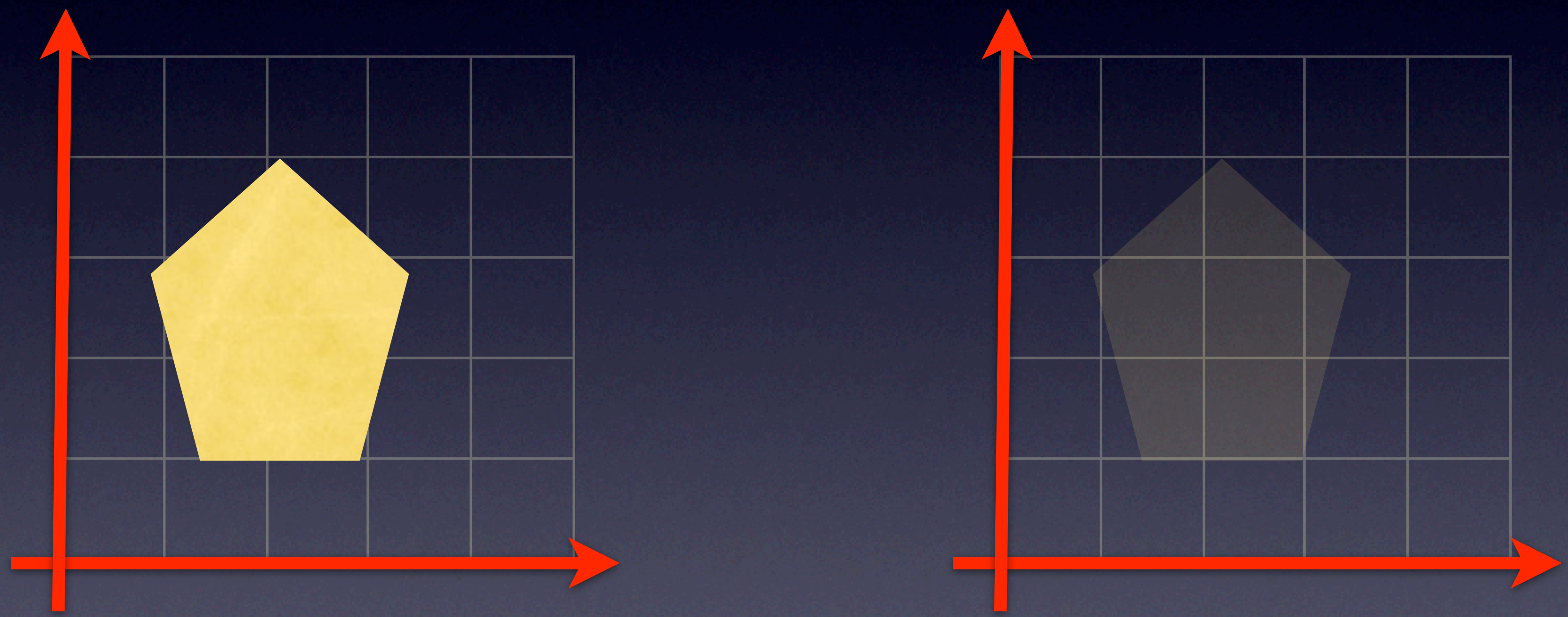
Translation



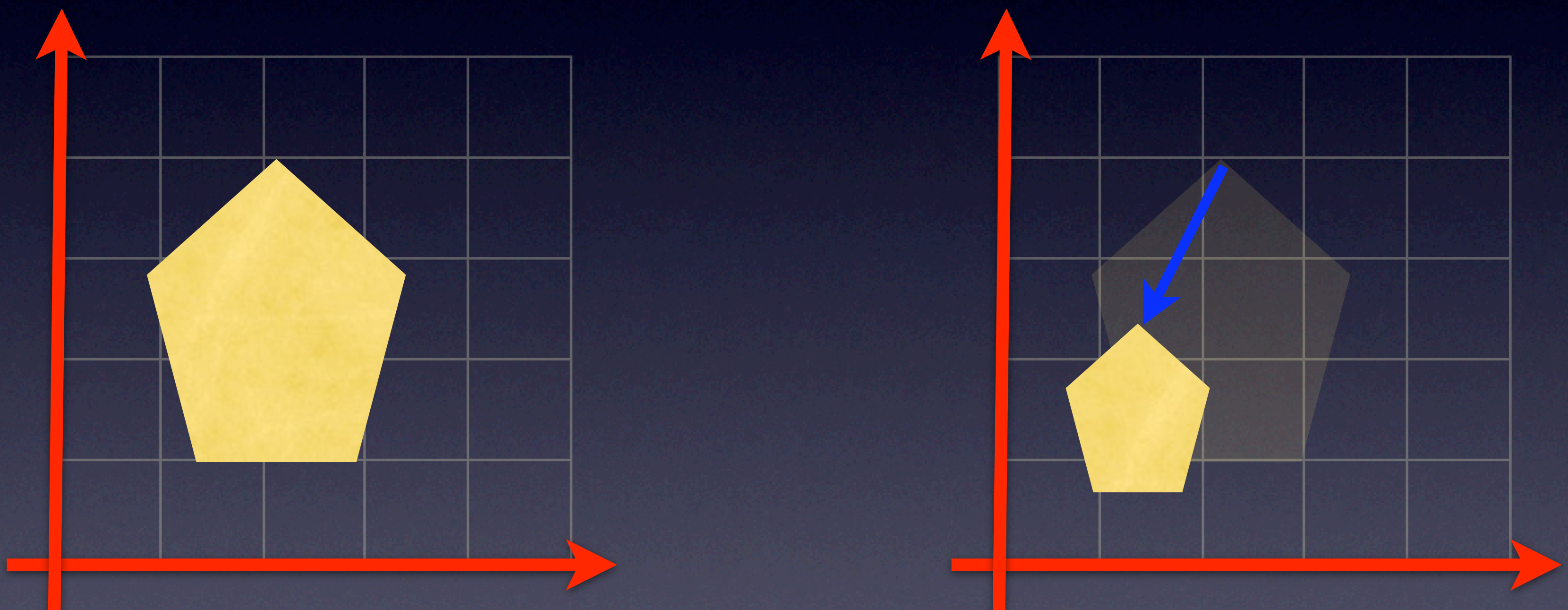
Translation



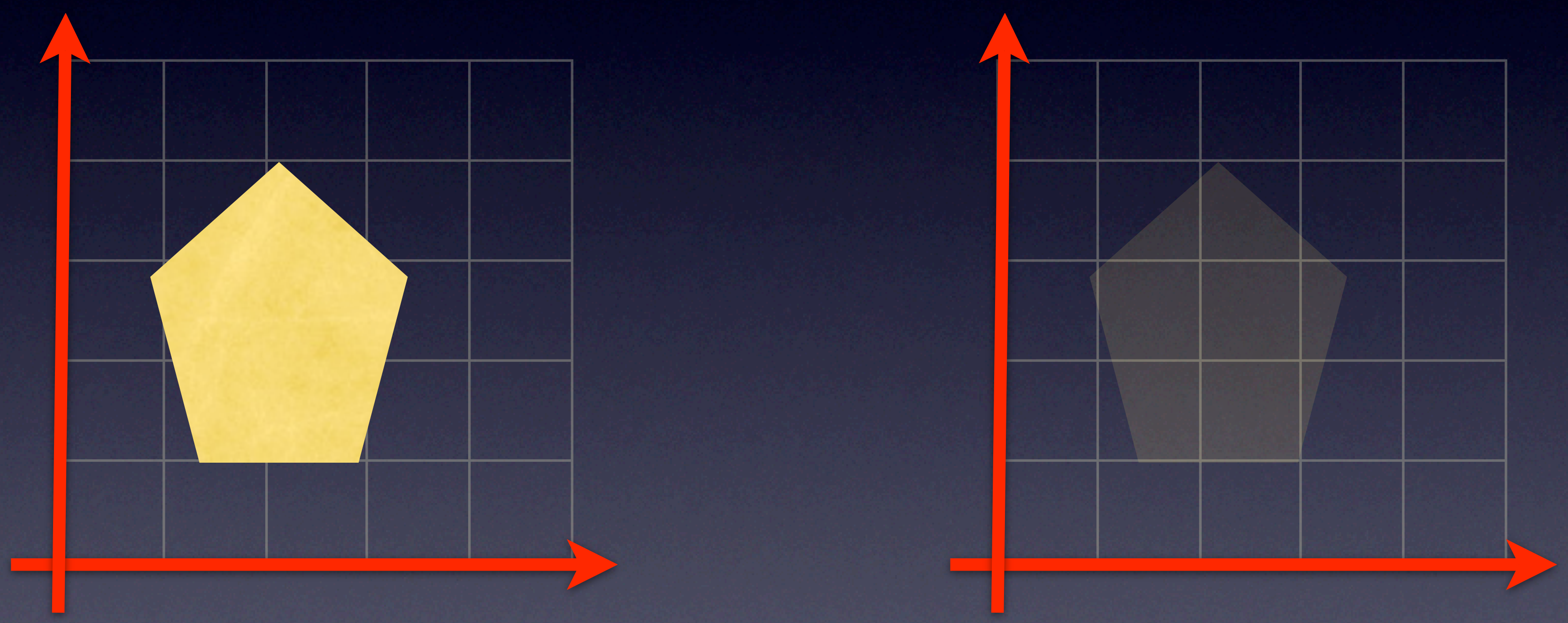
Uniform Scaling



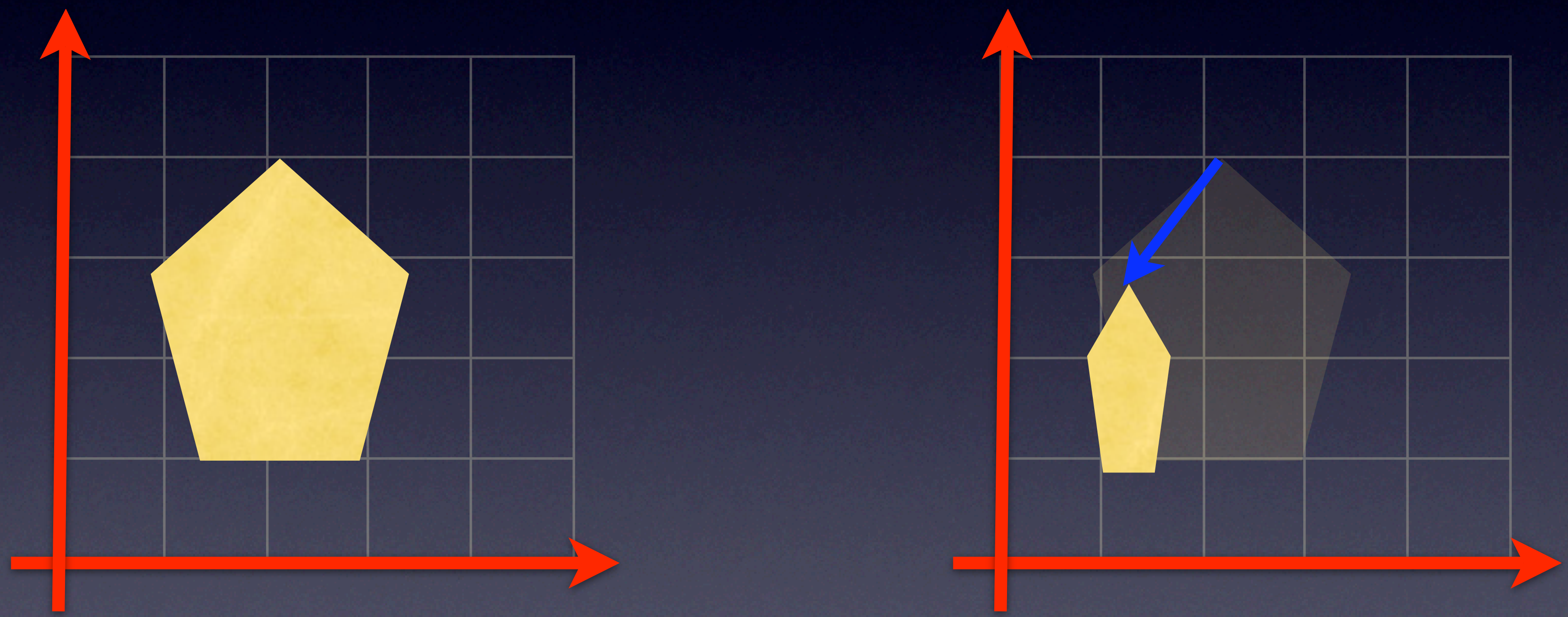
Uniform Scaling



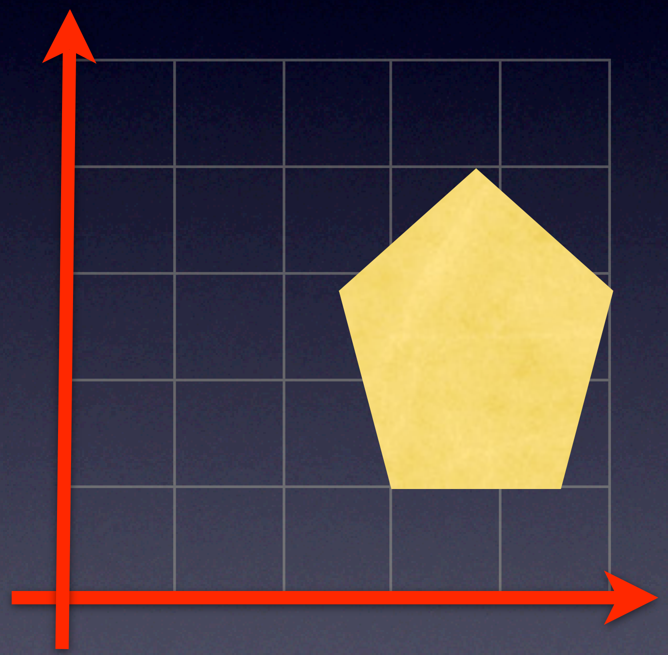
Non-uniform Scaling

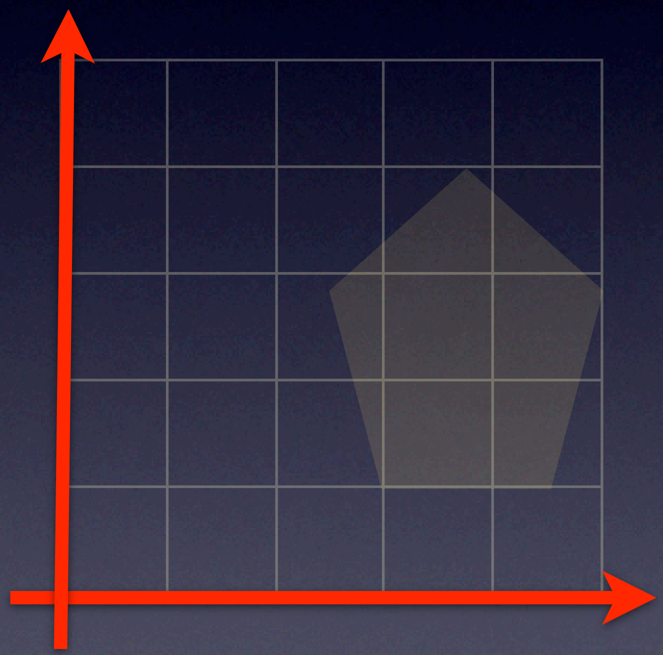


Non-uniform Scaling

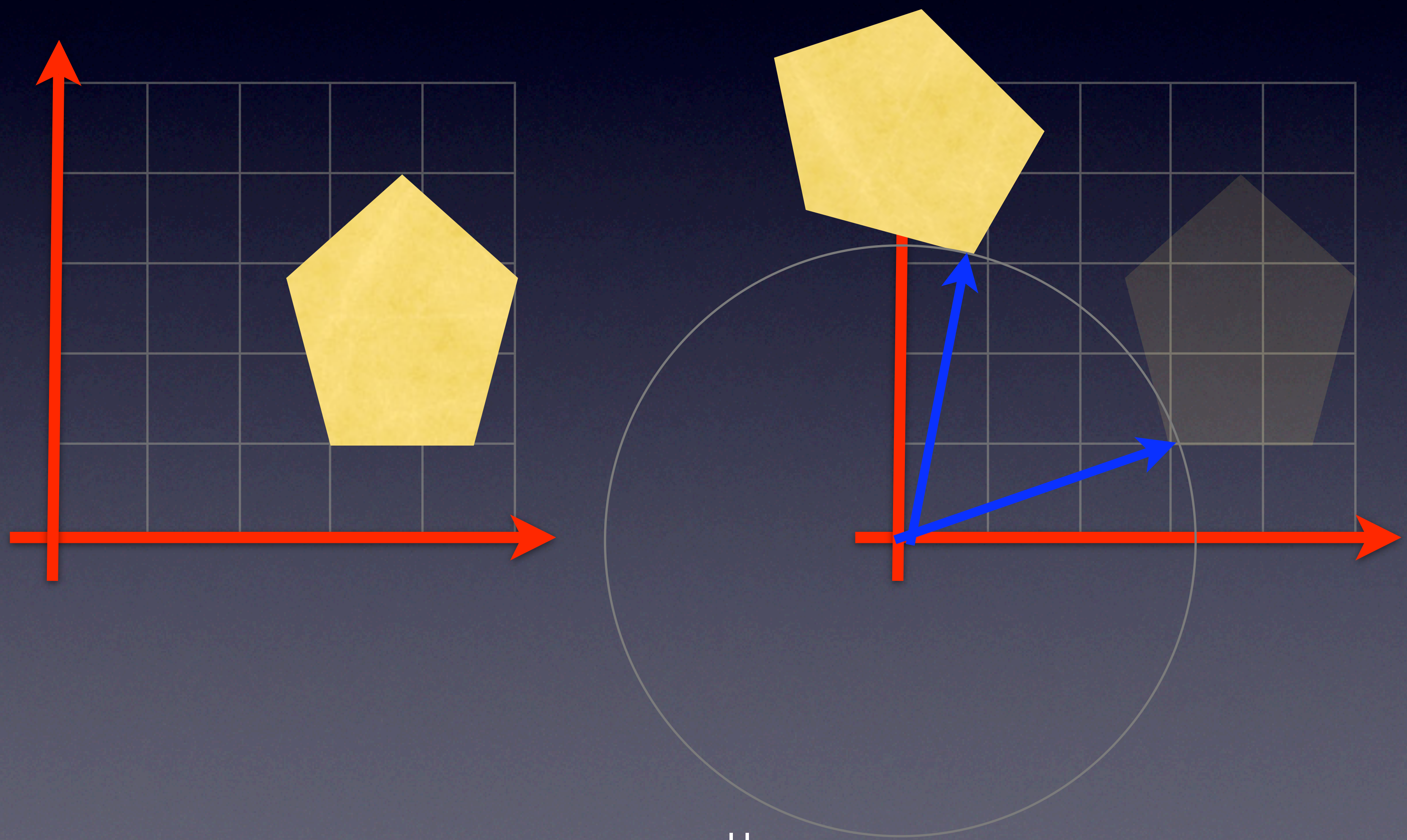


Rotation around origin

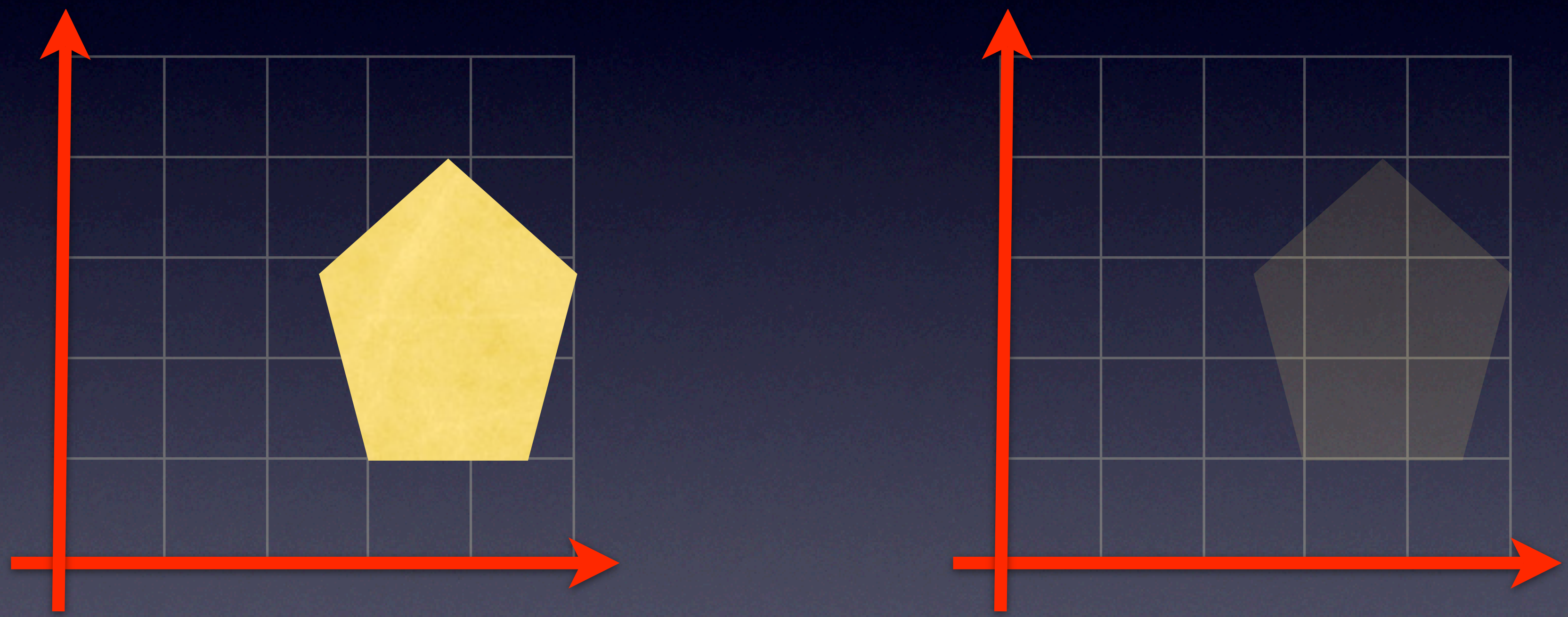




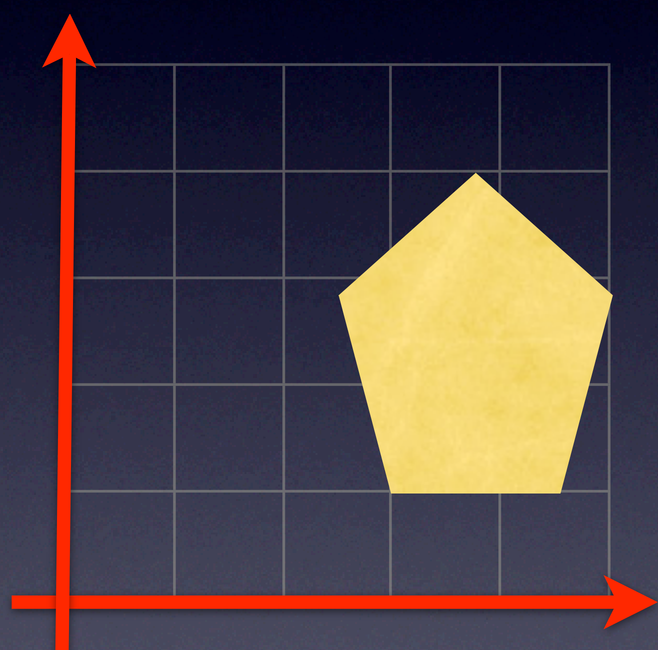
Rotation around origin

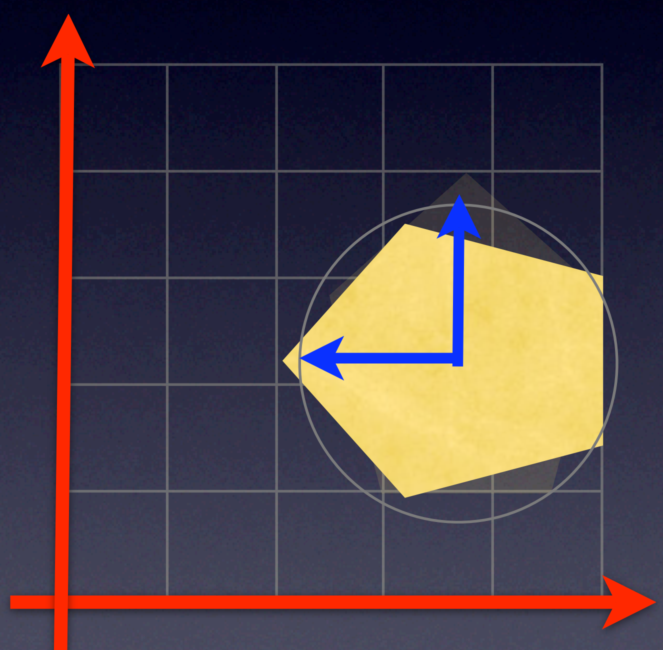


Rotation around center



Rotation around center





Translation

- Point P(X,Y) is to be translated by amount Dx and Dy to location P'(X',Y')
  - X' = Dx + X
  - Y' = Dy + Y
- P' = T + P where

$$P' = \begin{pmatrix} X' \\ Y' \end{pmatrix}, T = \begin{pmatrix} D_x \\ D_y \end{pmatrix}, P = \begin{pmatrix} X \\ Y \end{pmatrix}$$

Scaling

- Point P (X,Y) is to be scaled by amount Sx and Sy to location P'(X',Y')
  - X' = Sx * X
  - Y' = Sy * Y
- or P' = S * P where $S = \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix}$

Scaling

- Scaling is performed about the origin (0,0) not about the center of the primitive
- Scale > 1 enlarge the object and move it away from the origin.
  Scale = 1 leave the object alone
  Scale < 1 shrink the object and move it towards the origin.
- Uniform scaling Sx = Sy
- Differential scaling Sx != Sy
  - alters proportions

Rotation

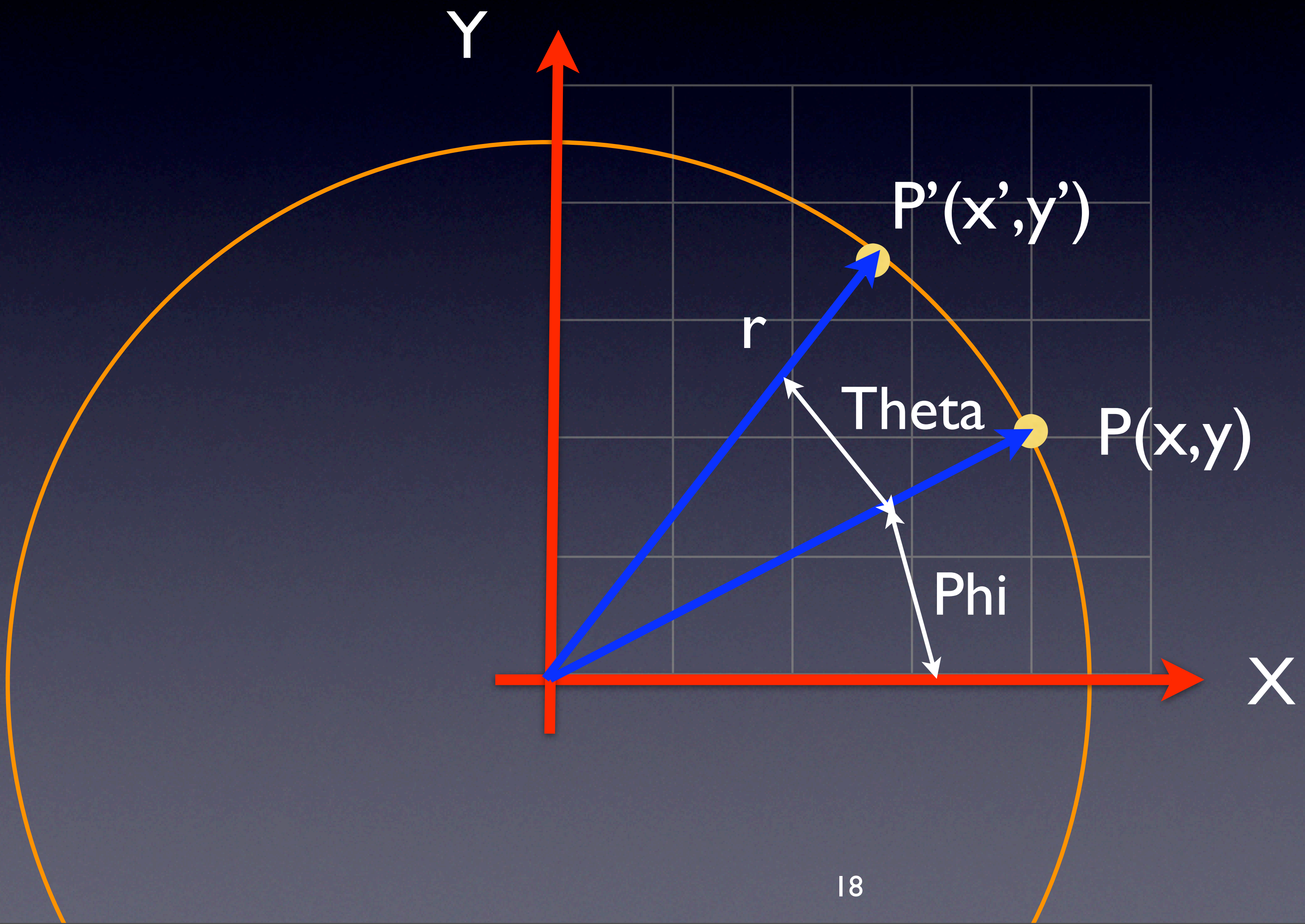
- Point (X,Y) is to be rotated about the origin by angle theta to location (X',Y')
  note that this does involve *sin* and *cos* which are much more costly than addition or multiplication
  - X' = X * cos(theta) - Y * sin(theta)
  - Y' = X * sin(theta) + Y *cos(theta)
- or P' = R * P where

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

Rotation

- Rotation is performed about the origin (0,0) not about the center of the line/polygon/whatever
- Where does this matrix come from?
  - (X,Y) is located r away from (0,0) at a CCW angle of phi from the X axis.
  - (X',Y') is located r away from (0,0) at a CCW angle of theta+phi from the X axis
- Since rotation is about the origin, (X',Y') must be the same distance from the origin as (X,Y)

Rotation



Rotation Matrix

- From trigonometry
  X = r * cos(phi)
  Y = r * sin(phi)
- X' = r * cos(theta+phi)
  Y' = r * sin(theta+phi)
- since
  cos(a+b) = cos(a) * cos(b) - sin(a) * sin(b)
  sin(a+b) = sin(a) * cos(b) + cos(a) * sin(b)
- X' = r * cos(theta) * cos(phi) - r * sin(theta) * sin(phi)
  Y' = r * sin(theta) * cos(phi) + r * cos(theta) * sin(phi)
- X' = X * cos(theta) - Y * sin(theta)
  Y' = X * sin(theta) + Y * cos(theta)

Operations

- Translation $P' = T + P$
- Scaling $P' = S * P$
- Rotation $P' = R * P$
- How to represent all operations as multiplication, in a consistent manner ?

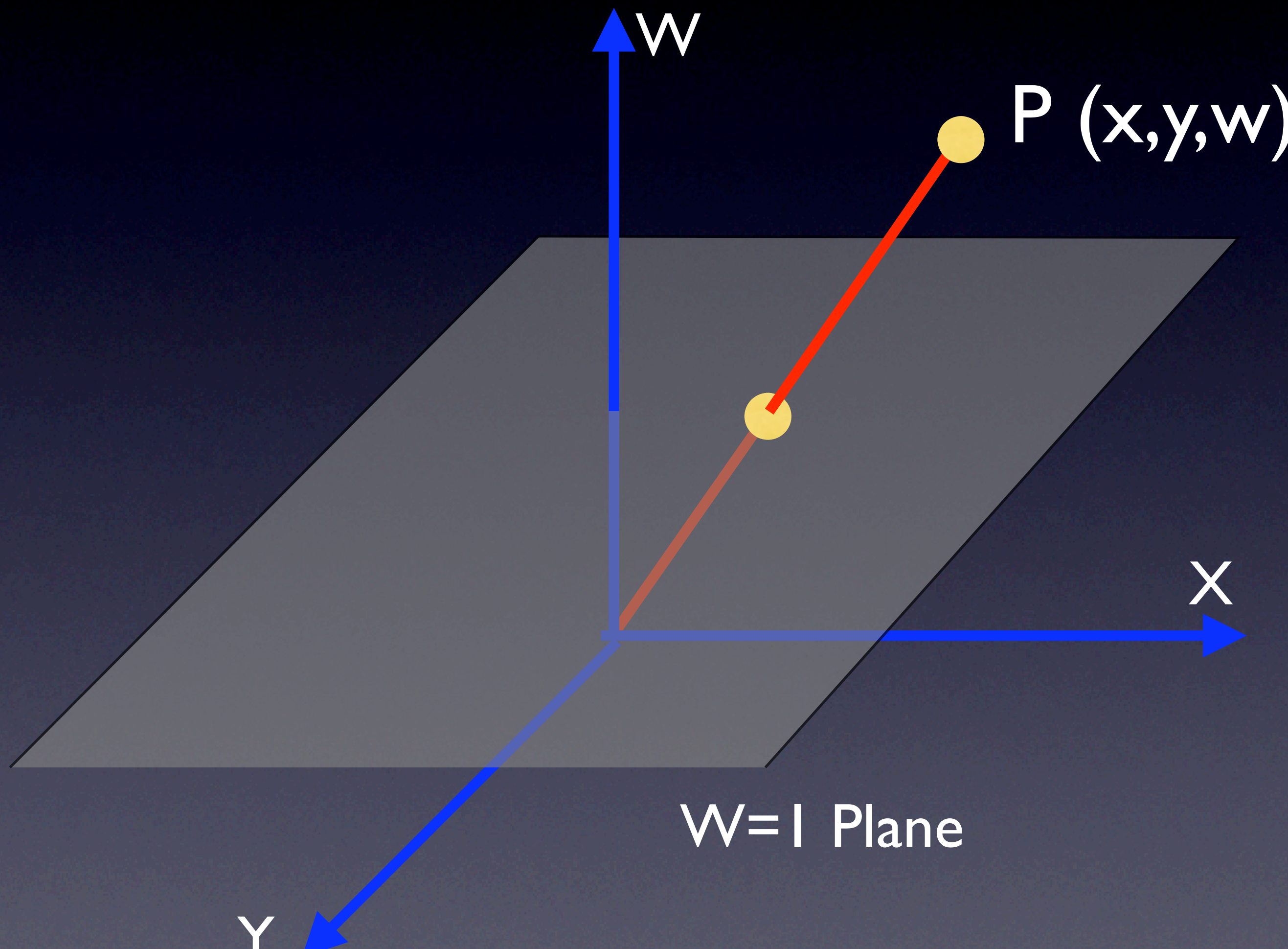
Homogeneous Coordinates

- Want to be able to treat all 3 transformations (translation, scaling, rotation) in the same way - as multiplications
- Each point given a third coordinate (X, Y, W)
- Two triples (X,Y,W) and (X',Y',W') represent the same point if they are multiples of each other
e.g. (1,2,3) and (2,4,6)
- At least one of the three coordinates must be nonzero

Homogeneous Coordinates

- If W is 0 then the point is at infinity
- If W is nonzero we can divide the triple by W to get the cartesian coordinates of X and Y
- Which will be identical for triples representing the same point (X/W, Y/W, 1)
- Homogeneous coordinates seem unintuitive, but they make graphics operations much easier

Homogeneous Coordinates



XYW homogeneous coordinate space

New Translation

- Point P(X,Y,1) is to be translated by amount Dx and Dy to location P’(X',Y',1)
  - X' = Dx + X
  - Y' = Dy + Y
- P' = T * P where
  - P’ = [X’ / Y’ /1]   P=[X / Y /1]

$$T = \begin{pmatrix} 1 & 0 & D_x \\ 0 & 1 & D_y \\ 0 & 0 & 1 \end{pmatrix}$$

New Scaling

- Point P (X,Y,1) is to be scaled by amount Sx and Sy to location P'(X',Y',1)
  - X' = Sx * X
  - Y' = Sy * Y
- or P' = S * P where
- P' = [X' / Y' / 1]   P=[X / Y / 1]

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

New Rotation

- Point (X,Y,1) is to be rotated about the origin by angle theta to location (X',Y',1)
  - X' = X * cos(theta) - Y * sin(theta)
  - Y' = X * sin(theta) + Y *cos(theta)
- or P' = R * P where

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

Composition of 2D Transformations

- Instead of applying several transformations matrices to a point, we want to use the transformations to produce 1 matrix which can be applied to the point
- In the simplest case, we want to apply the same type of transformation (translation, rotation, scaling) more than once

Composition

- Translation is additive as expected
- Scaling is multiplicative as expected
- Rotation is additive as expected
- But what if we want to combine different types of transformations?

Example

- A very common reason for doing this is to rotate a polygon about an arbitrary point (e.g. the center of the polygon) rather than around the origin
  - Translate so that P1 is at the origin T(-Dx,-Dy)
  - Rotate R(theta)
  - Translate so that the point at the origin is at P1 T(Dx,Dy)
- Order of operations here is right to left

$$P' = T\begin{pmatrix} D_x \\ D_y \end{pmatrix} * R\,(theta) * T\begin{pmatrix} -D_x \\ -D_y \end{pmatrix} * P$$

Another Example

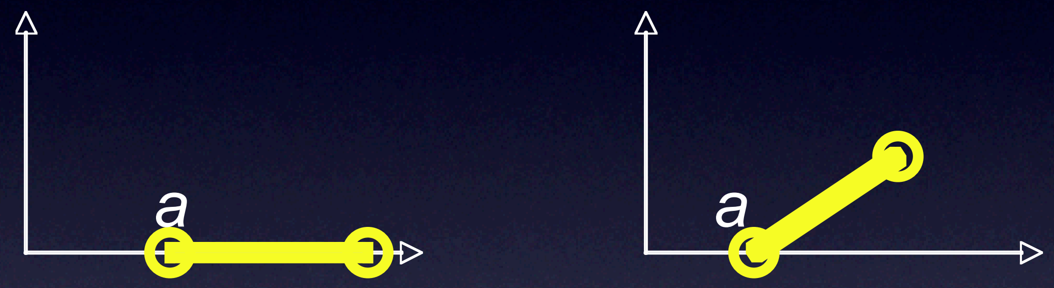
- Another common reason for doing this is to scale a polygon about an arbitrary point (e.g. the center of the polygon) rather than around the origin
  - Translate so that P1 is at the origin
  - Scale
  - Translate so that the point at the origin is at P1

Center of Polygon

- How do we determine the 'center' of the polygon?
  - Specifically define the center (e.g. the center of mass)
  - Average the location of all the vertices
  - Take the center of the bounding box of the polygon

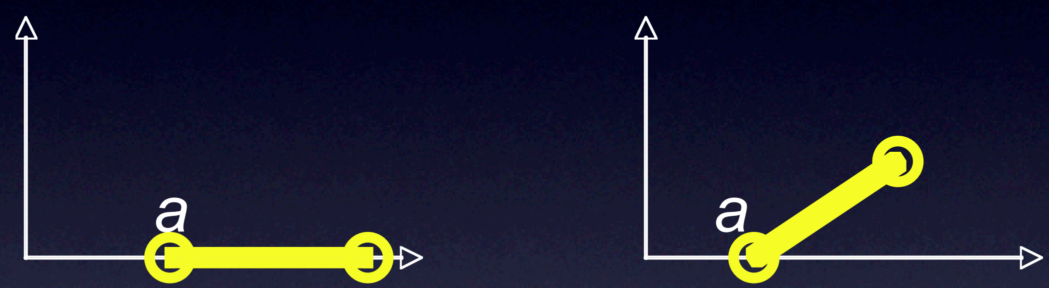
Example

Rotation around (a)

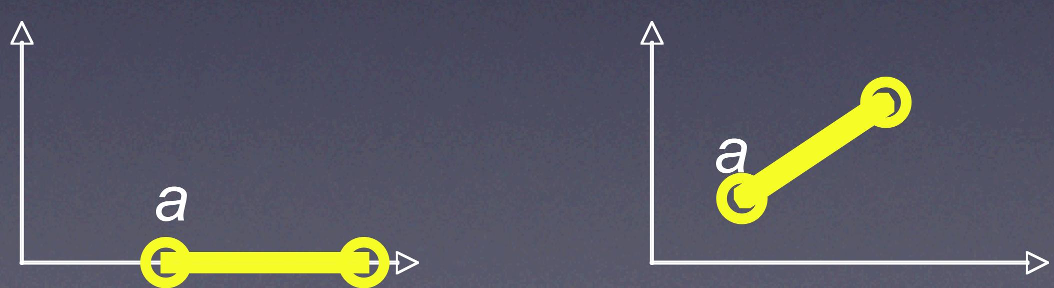


Example

Rotation around (a)

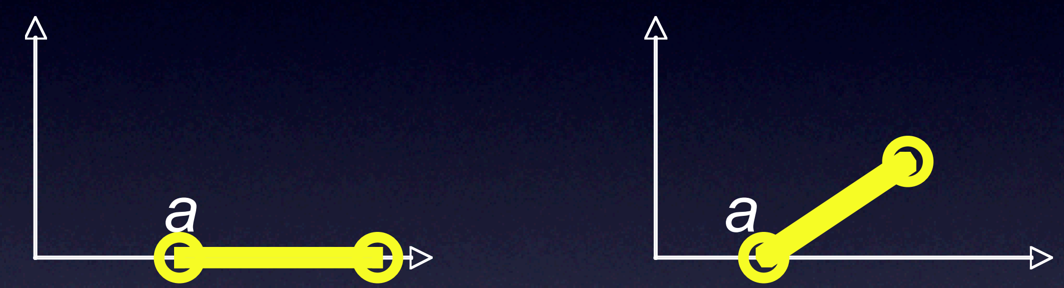


Applying rotation to segment →Wrong

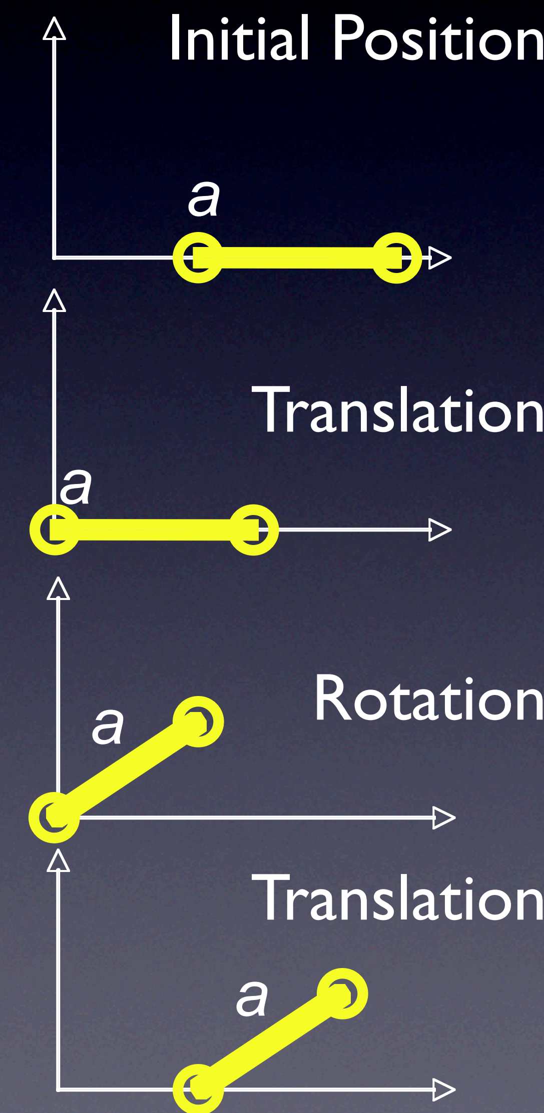


Example

Rotation around (a)



Applying rotation to segment →Wrong



Window to Viewport

- Generally user's prefer to work in world-coordinates.
  - 1 unit can be 1 micron
  - 1 unit can be 1 meter
  - 1 unit can be 1 kilometer
  - 1 unit can be 1 mile
- These coordinates must then be translated to screen coordinates to be displayed in a rectangular region of the screen called the viewport
- The objects are in world coordinates (with n dimensions)
  The viewport is in screen coordinates (with n=2)

Windows

- Want one matrix that can be applied to all points:
  - rectangular area of world from (Xmin, Ymin) to (Xmax, Ymax)
    - *world-coordinate window*
  - rectangular area of screen from (Umin, Vmin) to (Umax, Vmax)
    - *viewport*

Scaling back to screen

- Need to re-scale the world-coordinate rectangle to the screen rectangle
  1. Translate world-coordinate window to the origin of the world coordinate system.
  2. Re-scale the window to the size and aspect ratio of the viewport.
  3. Translate the viewport to its position on the screen in the screen coordinate system.
- Pscreen = M * Pworld

$$M = T\begin{pmatrix} U_{min} \\ V_{min} \end{pmatrix} * S\begin{pmatrix} deltaU/deltaX \\ deltaV/deltaY \end{pmatrix} * T\begin{pmatrix} -X_{min} \\ -Y_{min} \end{pmatrix}$$

3D Transformations

- Similar to 2D transformations, which used 3x3 matrices (X, Y, W)
- 3D transformations use 4X4 matrices (X, Y, Z, W)

3D Translation

- Point P (X,Y,Z,1) is to be translated by amount Dx, Dy and Dz to location (X',Y',Z',1)
  X' = Dx + X
  Y' = Dy + Y
  Z' = Dz + Z
- or P' = T * P where $T = \begin{pmatrix} 1 & 0 & 0 & D_x \\ 0 & 1 & 0 & D_y \\ 0 & 0 & 1 & D_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$

3D Scaling

- Point P (X,Y,Z,1) is to be scaled by amount Sx, Sy and Sz
  X' = Sx * X
  Y' = Sy * Y
  Z' = Sz * Z
- or P' = S * P where $S = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

3D Rotation

- We need to pick an axis to rotate about. The most common choices are the X-axis, the Y-axis, and the Z-axis
- Point P (X,Y,Z,1) to be rotated to P'(X',Y',Z',1) and angle *theta*

3D Rotations

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

$$P = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

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

$$P' = \begin{pmatrix} X' \\ Y' \\ Z' \\ 1 \end{pmatrix}$$

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

3D Composition

- Composition is handled in a similar way to the 2D case
- Multiplications of matrices

OpenGL Operations

- *glTranslate*{fd}(X, Y, Z)
  - glTranslatef(1.0, 2.5, 3.0)
- *glRotate*{df}(Angle, X, Y, Z)
  - glRotatef(60.0, 0.0, 0.0, 1.0)
- *glScale*{df}(X, Y, Z)
  - glScalef(1.0, 1.5, 2.0)

Next Time

- More Geometric Transformations