Student Online Teaching Advice Notice

The materials and content presented within this session are intended solely for use in a context of teaching and learning at Trinity.

Any session recorded for subsequent review is made available solely for the purpose of enhancing student learning.

Students should not edit or modify the recording in any way, nor disseminate it for use outside of a context of teaching and learning at Trinity.

Please be mindful of your physical environment and conscious of what may be captured by the device camera and microphone during videoconferencing calls.

Recorded materials will be handled in compliance with Trinity's statutory duties under the Universities Act, 1997 and in accordance with the University's policies and procedures.

Further information on data protection and best practice when using videoconferencing software is available at https://www.tcd.ie/info_compliance/data-protection/.

Geometric Transformations

Lecturer: Carol O’Sullivan

Credits: Some slides taken from Robb T. Koether, Hampden-Sydney College

Objectives

- Learn how to carry out transformations
  - Rotation
  - Translation
  - Scaling
  - Combinations!

Computer Graphics Problems

- Much of graphics concerns itself with the problem of displaying 3D objects in 2D screen
- We want to be able to:
  - rotate, translate, scale our objects
  - view them from arbitrary points of view
  - View them in perspective
- Want to display objects in coordinate systems that are convenient for us and to be able to reuse object descriptions
- Road example
  - Cars, tyres
  - View from a helicopter

Matrices

- If you need to rotate a million vertices representing a dinosaur object about some axis, you don’t need to multiply each point by 5 different matrices
  - you simply multiply the 5 matrices together once and multiply each dinosaur point by that one matrix. Huge saving!

Matrices

- Matrix addition

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} + \begin{bmatrix} e & g \\ f & h \end{bmatrix} = \begin{bmatrix} a+e & c+g \\ b+f & d+h \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 \\ 3 & 2 \end{bmatrix} + \begin{bmatrix} 4 & 1 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 6 & 2 \\ 5 & 6 \end{bmatrix}$$

Matrices

- Matrix multiplication

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} \text{x} \begin{bmatrix} e & g \\ f & h \end{bmatrix} = \begin{bmatrix} \boxed{ae+cf} & ag+ch \\ be+df & bg+dh \end{bmatrix}$$

Matrices

- Matrix multiplication

$$\begin{bmatrix} \boxed{a \quad c} \\ b \quad d \end{bmatrix} \mathrm{x} \begin{bmatrix} e & \boxed{\begin{matrix} g \\ h \end{matrix}} \\ f & \end{bmatrix} = \begin{bmatrix} ae + cf & \boxed{ag + ch} \\ be + df & bg + dh \end{bmatrix}$$

Matrices

- Matrix multiplication

$$\begin{bmatrix} a & c \\ \boxed{b \quad d} \end{bmatrix} \text{x} \begin{bmatrix} \boxed{e} & g \\ \boxed{f} & h \end{bmatrix} = \begin{bmatrix} ae + cf & ag + ch \\ \boxed{be + df} & bg + dh \end{bmatrix}$$

Matrices

- Matrix multiplication

$$\begin{bmatrix} a & c \\ \boxed{b \quad d} \end{bmatrix} \text{x} \begin{bmatrix} e & \boxed{g} \\ f & \boxed{h} \end{bmatrix} = \begin{bmatrix} ae + cf & ag + ch \\ be + df & \boxed{bg + dh} \end{bmatrix}$$

Matrices

- Matrix multiplication not commutative in most cases
  - **AB** /= **BA**
  - If **AB** = **AC**, it does not necessarily follow that **B** = **C**
- It is associative and distributive
  - (AB)C = A(BC)
  - A(B+C) = AB + AC
  - (A+B)C = AC + BC

$$A = r^2$$

- Transpose A^T of a matrix A is one whose rows are switched with its columns

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}^T$$

Geometric Transformations

- Many geometric transformations are *linear* and can be represented as a matrix multiplication.
- Function *f* is linear iff:

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$$

- <u>Implications</u>:
  - to transform a line we transform the *end-points*. Points between are *affine combinations* of the transformed endpoints.
  - Given line defined by points *P* and Q, points along transformed line are affine combinations of transformed *P′* and Q′

$$L(t) = P + t(Q - P)$$
$$L'(t) = P' + t(Q' - P')$$

Homogeneous Co-ordinates

- Basis of the homogeneous co-ordinate system is the set of *n* basis vectors and the origin position:

$$\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n \text{ and } P_o$$

- All points and vectors are therefore compactly represented using their ordinates:

$$\begin{bmatrix} a_1 \\ \vdots \\ a_n \\ a_o \end{bmatrix} \text{or more usually} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Homogeneous Co-ordinates

- Vectors have no positional information and are represented using $a_o = 0$ whereas points are represented with $a_o = 1$:

$$\vec{v} = a_1\mathbf{v}_1 + \cdots + a_n\mathbf{v}_n + 0$$

$$P = a_1\mathbf{v}_1 + \cdots + a_n\mathbf{v}_n + P_o$$

- <u>Examples</u>:

$$\begin{bmatrix} 0.2 \\ 1.3 \\ 2.2 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1.0 \\ 1.0 \\ 0.0 \\ 1 \end{bmatrix} \qquad\qquad \begin{bmatrix} 0.2 \\ 1.3 \\ 2.2 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1.0 \\ 1.0 \\ 0.0 \\ 0 \end{bmatrix}$$

Points — Associated vectors

4D Vectors

- XYZ and W
- vec4 in GLSL
- For POINTS, set the 4th component to 1.0
- For VECTORS, set the 4th component to 0.0
- Q: Any idea why?

- vec4 (1.0, 5.0, -10.0, 0.0);
- vec4 (1.0, 5.0, -10.0, 1.0);

Homogenous Coordinates

- Using this scheme, every rotation, translation, and scaling operation can be represented by a matrix multiplication, and any combination of the operations corresponds to the products of the corresponding matrices
- Using homogeneous co-ordinates allows us to treat translation in the same way as rotation and scaling

Translation

- Simplest of the operations
  - Add a positive number – moves to the right
  - Add a negative number – moves to the left
- Addition of constant values, causes uniform translations in those directions
- Translations are independent and can be performed in any order (including all at once)
  - Object moved one unit to the right then up
  - Same as if moved one unit up and to the right
  - Net result is motion of sqrt(2) units to the upper-right

Translation

Definition (Translation)

A translation is a displacement in a particular direction

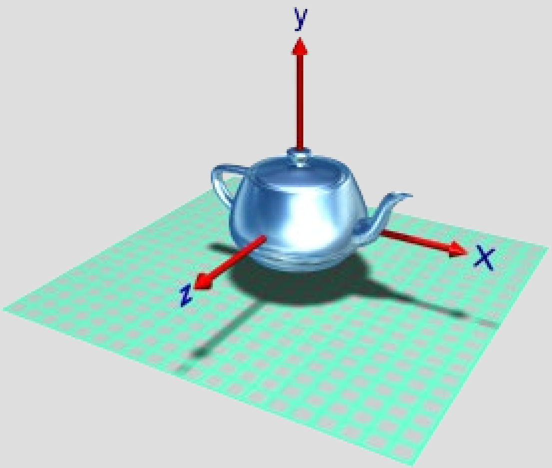
- A translation is defined by specifying the displacements a, b, and c

$$x' = x + a$$
$$y' = y + b$$
$$z' = z + c$$

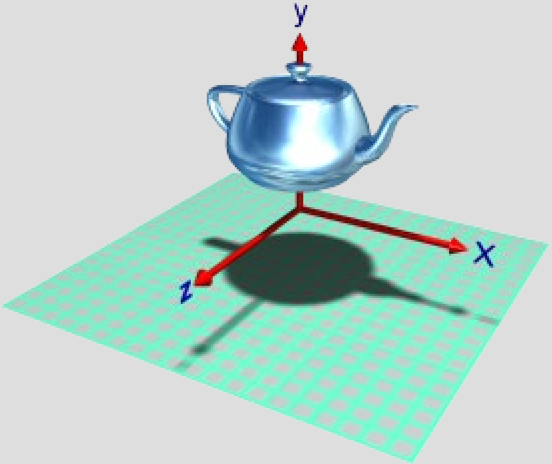
Translation

- Translation *only applies to points*, we never translate vectors.
- <u>Remember</u>: points have homogeneous co-ordinate *w* = *1*

$$\begin{aligned} x' &= x + a \\ y' &= y + b \\ z' &= z + c \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \\ z+c \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



translate along ***y***

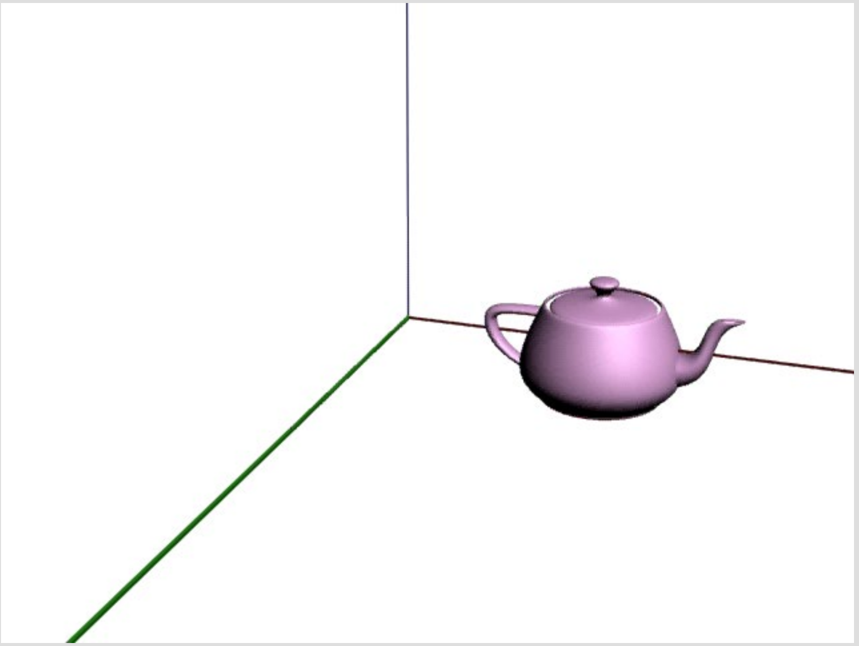


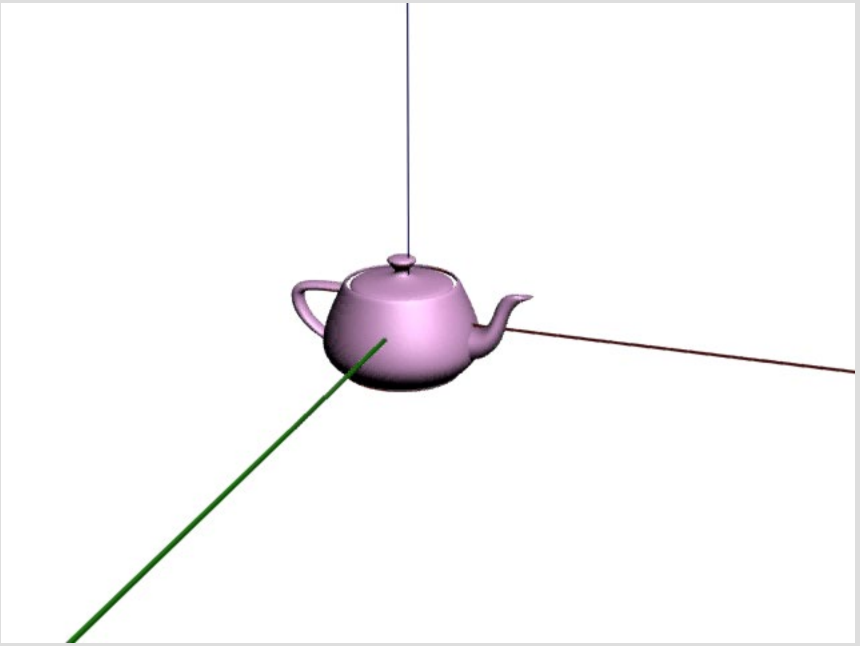
Scaling

- What if we want to make things larger or smaller?
- Have a car model
  - Want one 3 times smaller!

Scaling an object

- 3 times smaller
- Multiply all our coordinates by 1/3
- We get a model that is 1/3 of the size
- However
  - If original coordinates described a car 1 mile from the origin
  - Miniature car would only be 1/3 mile from the origin
- Solution – translation to origin, and then scale, then translate back





Scaling

Definition (Scaling)

A scaling is an expansion or contraction in the x, y, and z directions by scale factors sx, sy, sz and centred at the point (a,b, c)

- Generally we centre the scaling at the origin

$$x' = s_x x$$

$$y' = s_y y$$

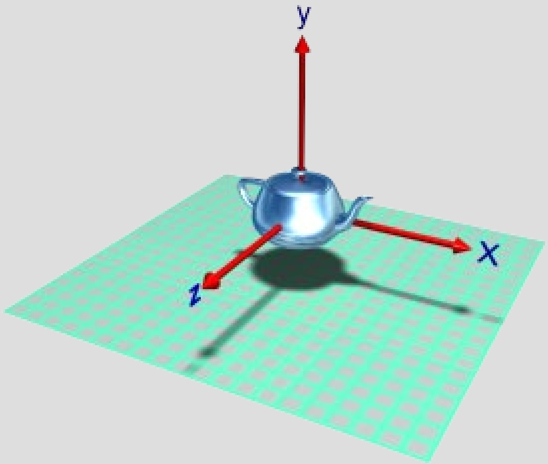
$$z' = s_z z$$

Non-Uniform Scaling

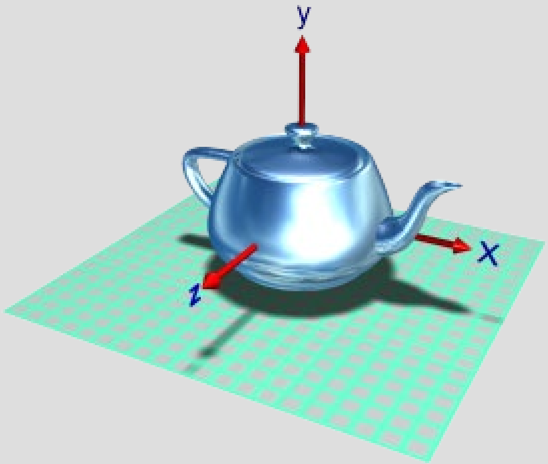
- Make an object twice as big in the x-direction
  - Multiply all x-coordinates by 2, leave y&z unchanged
- 3 times as large in the y-direction
  - Multiply all y-coordinates by 3, leave z&x unchanged

Scale

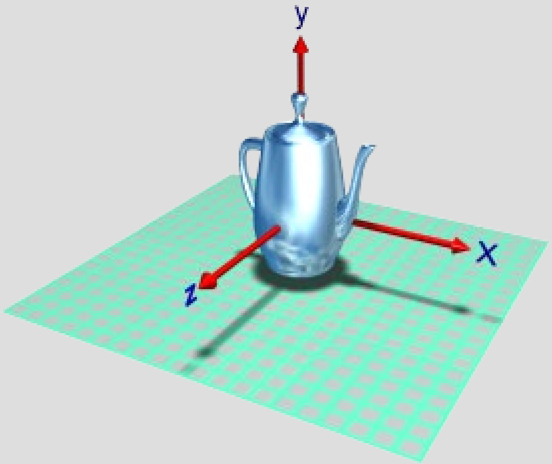
- all vectors are scaled from the origin:



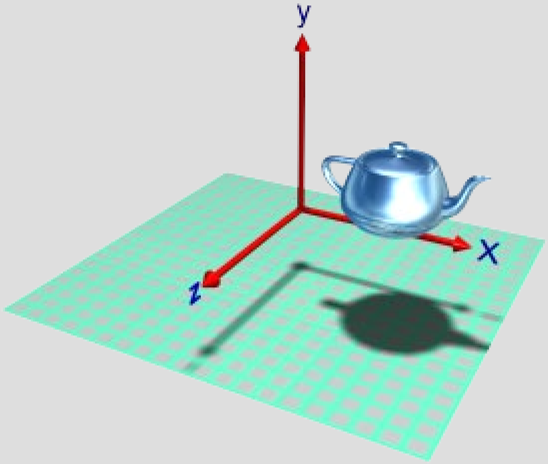
Original



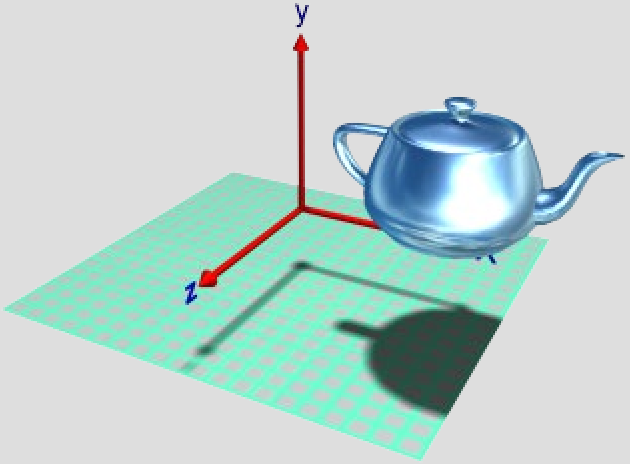
scale all axes



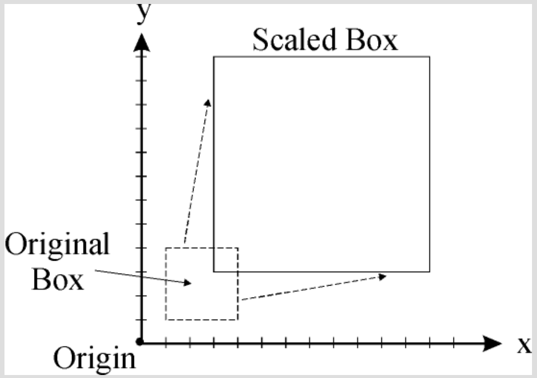
scale Y axis



offset from origin



distance from origin also scales



Scale

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \Rightarrow \quad \mathbf{v}' = \mathbf{S}\mathbf{v}$$

We would also like to scale points thus we need a *homogeneous transformation* for consistency:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \\ w \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \qquad \mathbf{S}^{-1} = \begin{bmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

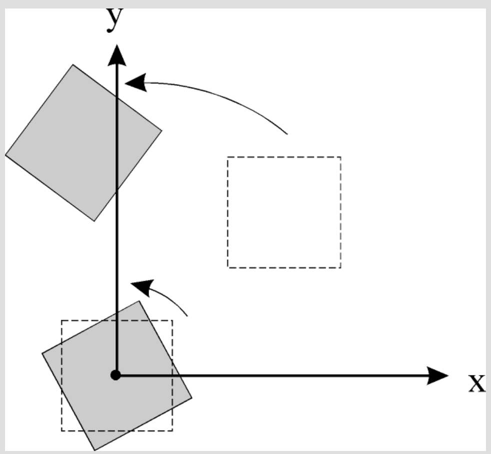
Rotation

Definition (Rotation)

A rotation turns about a point (a,b) through an angle θ

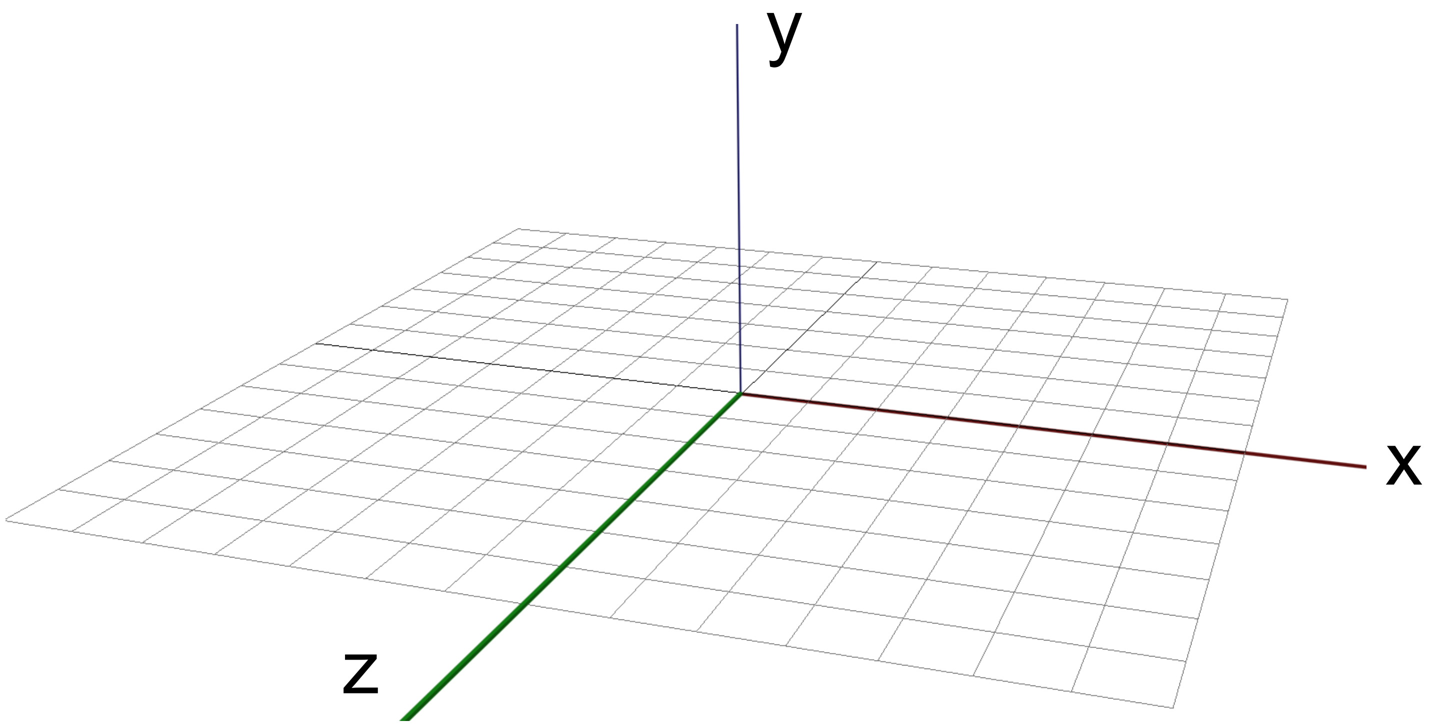
- Generally, we rotate about the origin
- Using the z-axis as the axis of rotation, the equations are:

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$



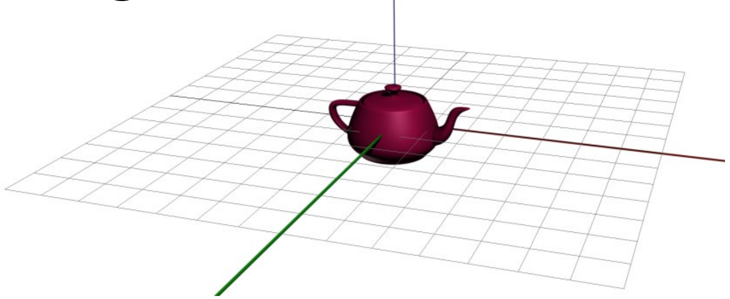
Rotation - idea

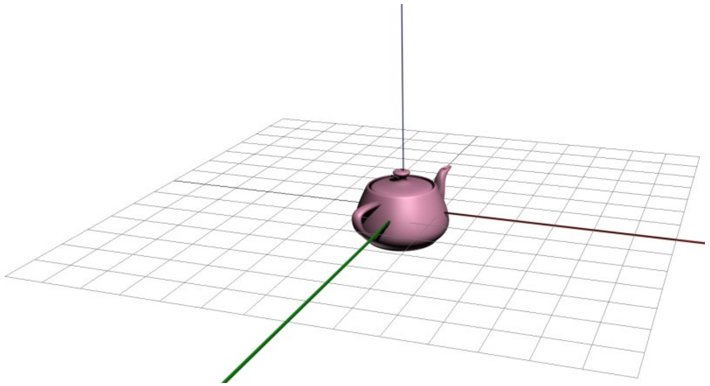
- Visualise rotation about an axis:
  - Put your eye on that axis in the positive direction and look towards the origin
  - Then, a positive rotation corresponds to a counter-clockwise rotation

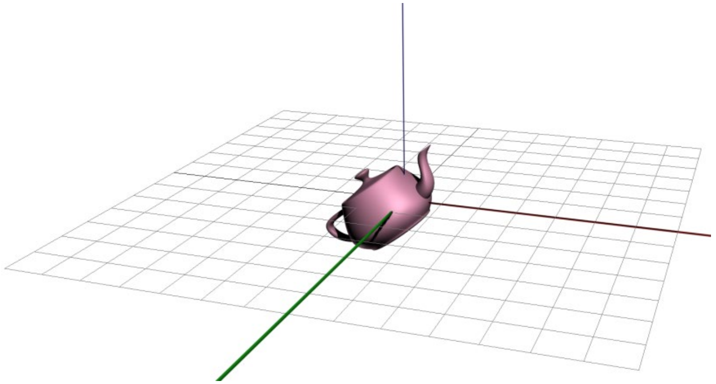


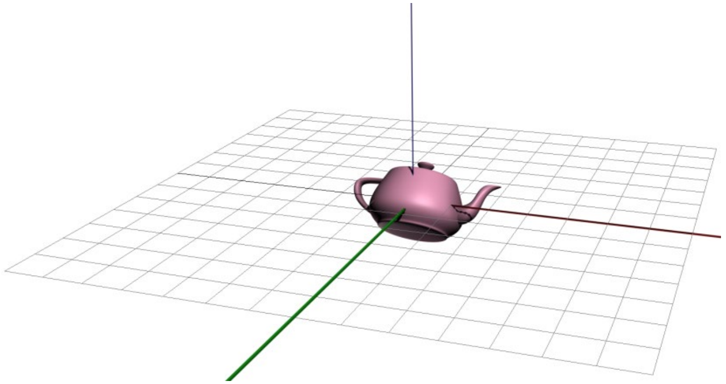
Which Axis?

original



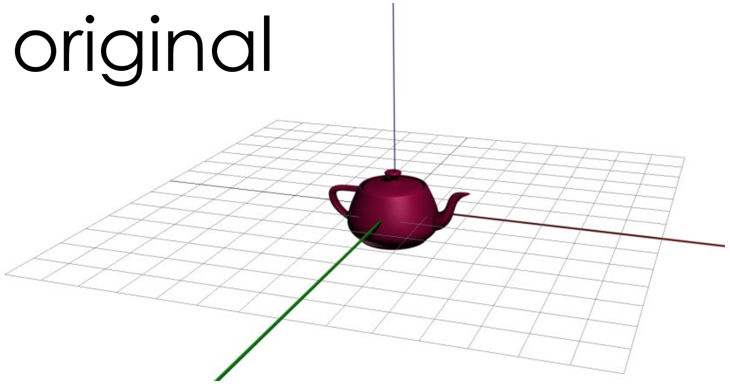




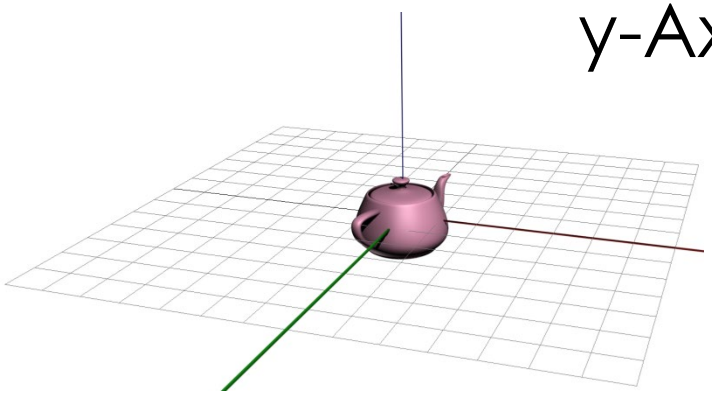


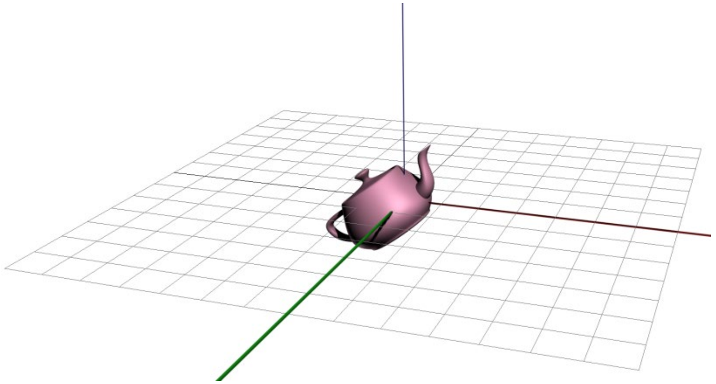
Which Axis?

original

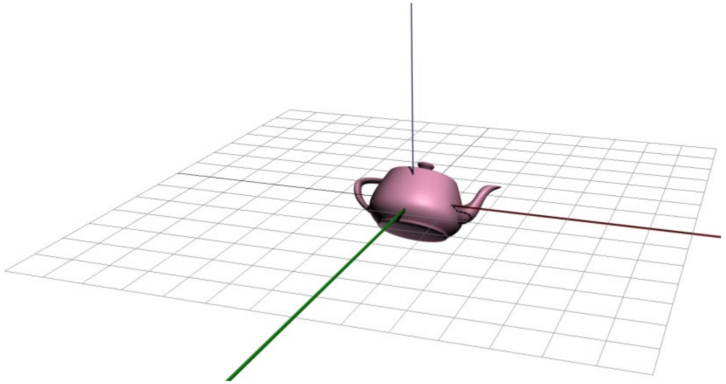


y-Axis





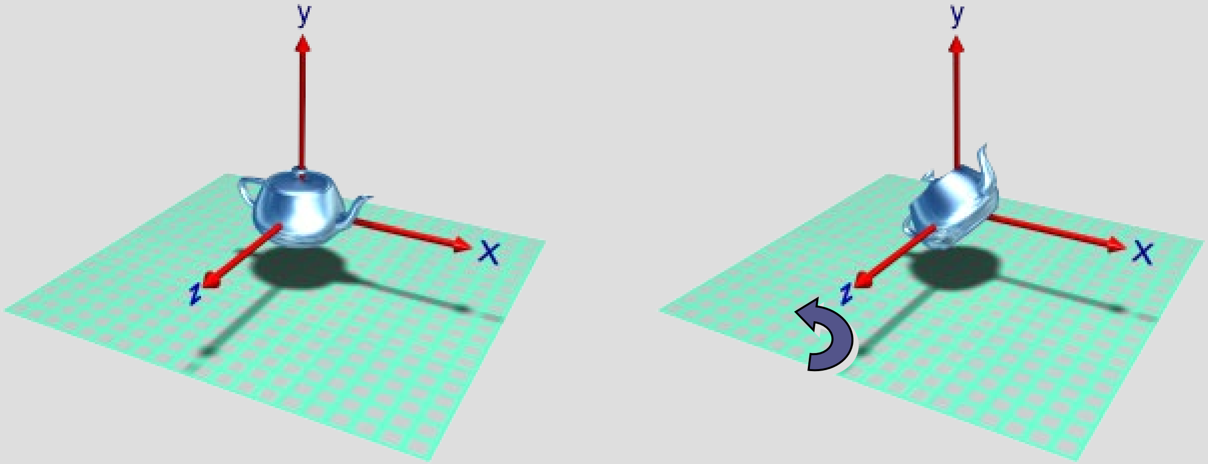
z-axis



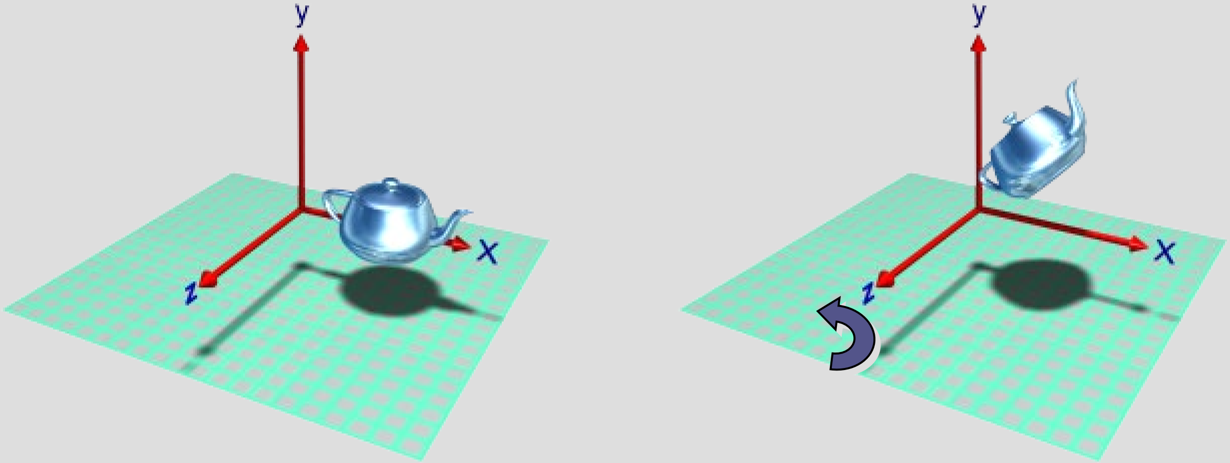
(-) x-axis

Rotation

- Rotations are *anti-clockwise* about the *origin:*



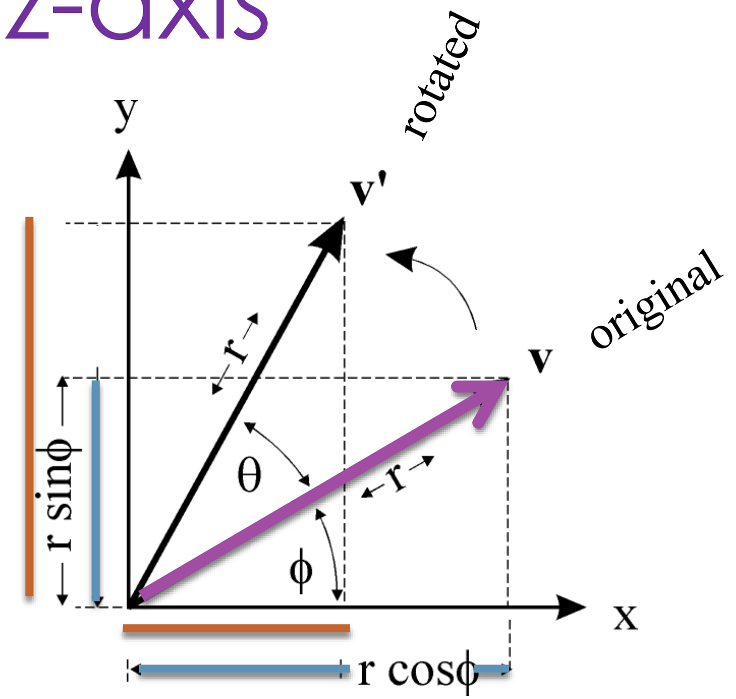
rotation of 45^o about the Z axis



offset from origin rotation

Rotation about the z-axis

$$\mathbf{v} = \begin{bmatrix} r\cos\phi \\ r\sin\phi \end{bmatrix} \qquad \mathbf{v}' = \begin{bmatrix} r\cos(\phi+\theta) \\ r\sin(\phi+\theta) \end{bmatrix}$$



$$\text{expand } (\phi+\theta) \Rightarrow \begin{cases} x' = r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ y' = r\cos\phi\sin\theta + r\sin\phi\cos\theta \end{cases}$$

$$\text{but} \quad \begin{array}{l} x = r\cos\phi \\ y = r\sin\phi \end{array} \Rightarrow \begin{array}{l} x' = x\cos\theta - y\sin\theta \\ y' = x\sin\theta + y\cos\theta \end{array}$$

Rotation

- 2D rotation of θ about origin: $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$
- 3D homogeneous rotations:

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- <u>Note</u>: $\begin{matrix} \cos(-\theta) = \cos\theta \\ \sin(-\theta) = -\sin\theta \end{matrix} \Rightarrow \mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta) = \mathbf{R}^{T}(\theta)$
- If $\mathbf{M}^{-1} = \mathbf{M}^{T}$ then **M** is *orthonormal*. All orthonormal matrices are rotations about the origin.

Which axis is this a rotation around?

Model (World) Matrix

1.00	0.00	0.00	0.00
0.00	-0.83	0.56	0.00
0.00	-0.56	-0.83	0.00
0.00	0.00	0.00	1.00

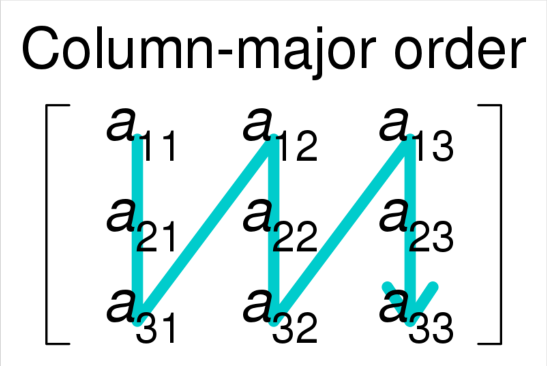
$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Answer

So it’s a rotation around the X axis, as the first row multiplied by the vertex keeps the x value the same

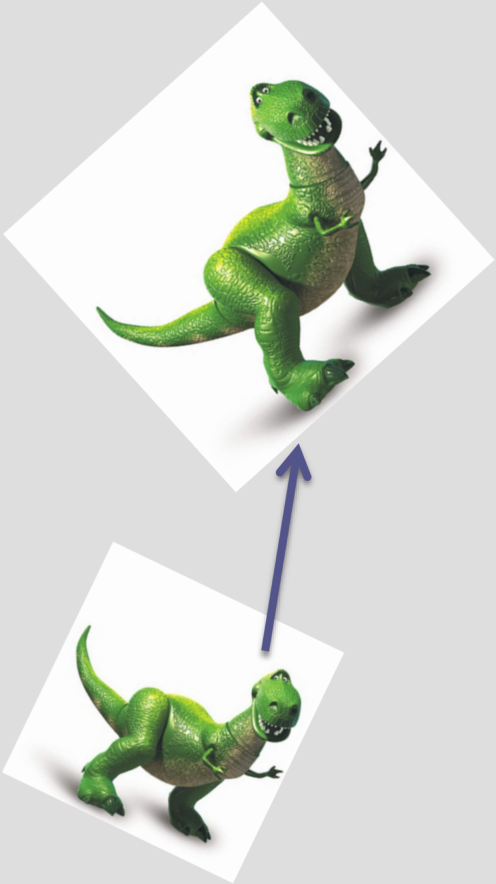
Vertex Shader for Rotation

```
// Remember: these matrices are column-major*
(unlike typical c-programming array filling)

    mat4 rx = mat4( 1.0,  0.0,  0.0, 0.0,
                    0.0,  c.x,  s.x, 0.0,
                    0.0, -s.x,  c.x, 0.0,
                    0.0,  0.0,  0.0, 1.0 );

     mat4 ry = mat4( c.y, 0.0, -s.y, 0.0,
                     0.0, 1.0,  0.0, 0.0,
                     s.y, 0.0,  c.y, 0.0,
                     0.0, 0.0,  0.0, 1.0 );

//note – theta will be in radians in C
//Right-hand rule for rotation directions
//glUniformMatrix4v – set flag to “false”
```



Combining Rotation, Translation, & Scaling

- Often advantageous to combine various transformations to form a more complex transformation
- If we do the algebra – things get complicated quickly
- Easier method – matrices

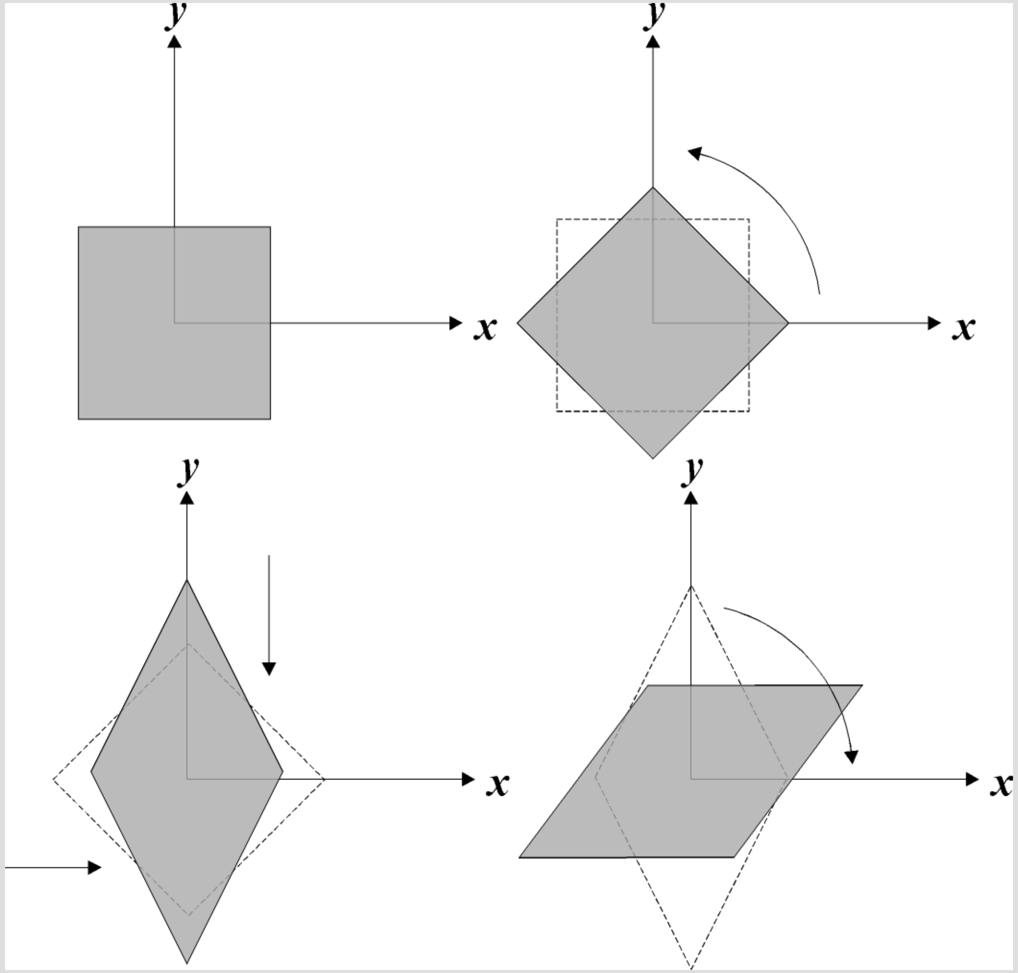


Homogenous Coordinates

- Using this scheme, every rotation, translation, and scaling operation can be represented by a matrix multiplication, and
- any combination of the operations corresponds to the products of the corresponding matrices

Affine Transformations

- All *affine transformations* are combinations of rotations, scaling and translations.



Transformation Composition

- It is common for graphics programs to apply more than one transformation to an object
  - Take vector v_1, Scale it (**S**), then rotate it (**R**)
  - First, $v_2 = \mathbf{S}v_1$, then, $v_3 = \mathbf{R}v_2$
  - $v_3 = \mathbf{R}(\mathbf{S}v_1)$
  - Since matrix multiplication is associative: $v_3 = (\mathbf{RS})v_1$
- In other words, we can represent the effects of transforms by two matrices in a single matrix of the same size by multiplying the two matrices: $\mathbf{M} = \mathbf{RS}$

Transformation Composition

- More complex transformations can be created by *concatenating* or *composing* individual transformations together.

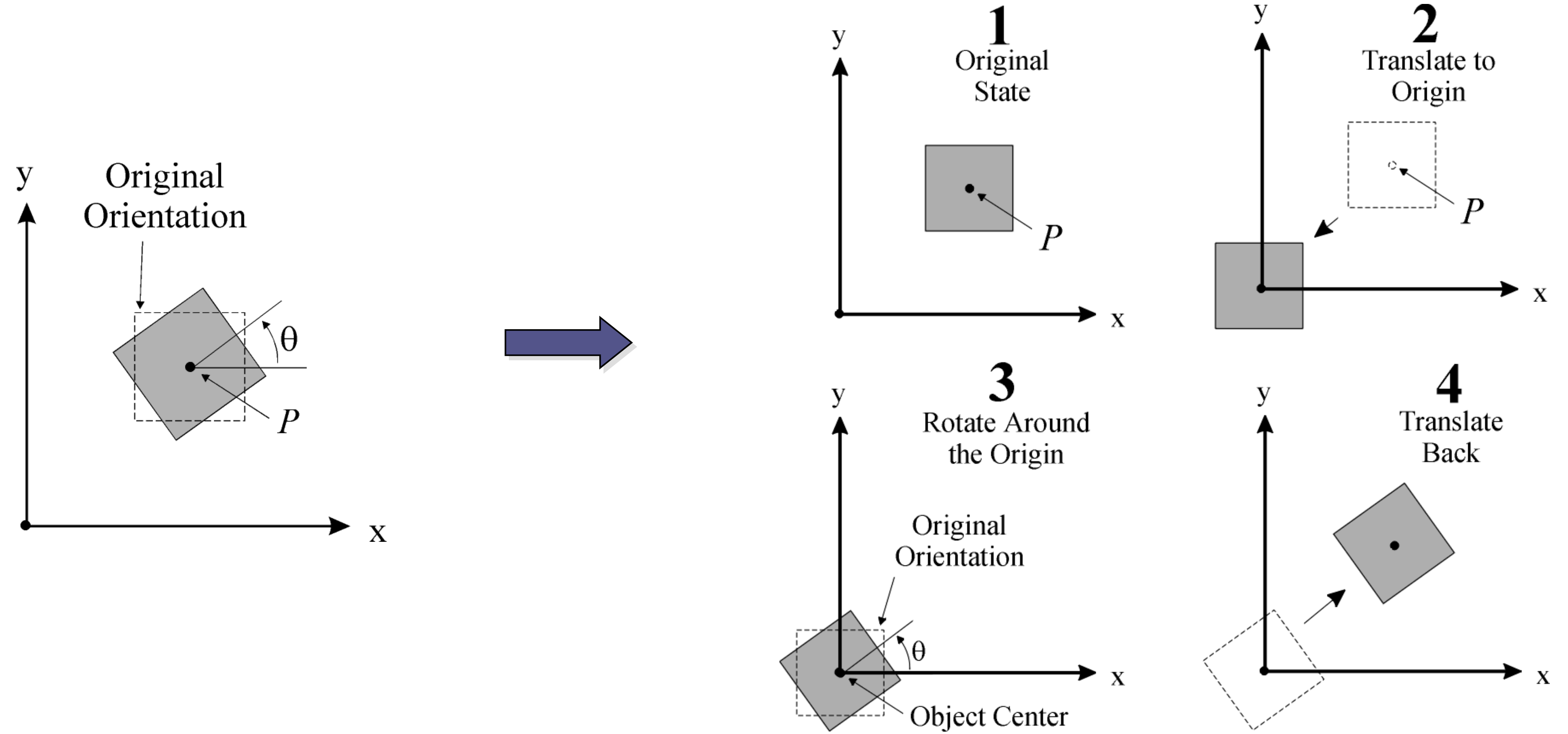
$$\mathbf{M} = \mathbf{T} \circ \mathbf{R} \circ \mathbf{S} \circ \mathbf{T} = \mathbf{TRST} \qquad \mathbf{v}' = \mathbf{T}[\mathbf{R}[\mathbf{S}[\mathbf{Tv}]]] = \mathbf{Mv}$$

- Matrix multiplication is *non-commutative* $\Rightarrow$ **order is vital**
- We can create an affine transformation representing rotation about a point P_R:

= *translate to origin, rotate about origin, translate back to original location*

$$\mathbf{M} = \mathbf{T}(P_R)\mathbf{R}(\theta)\mathbf{T}(-P_R)$$

Rotation about a point



Transformation Composition

Rotation in **XY** plane by *q* degrees anti-clockwise about point *P*

$$\mathbf{M} = \mathbf{T}(P)\mathbf{R}(\theta)\mathbf{T}(-P)$$

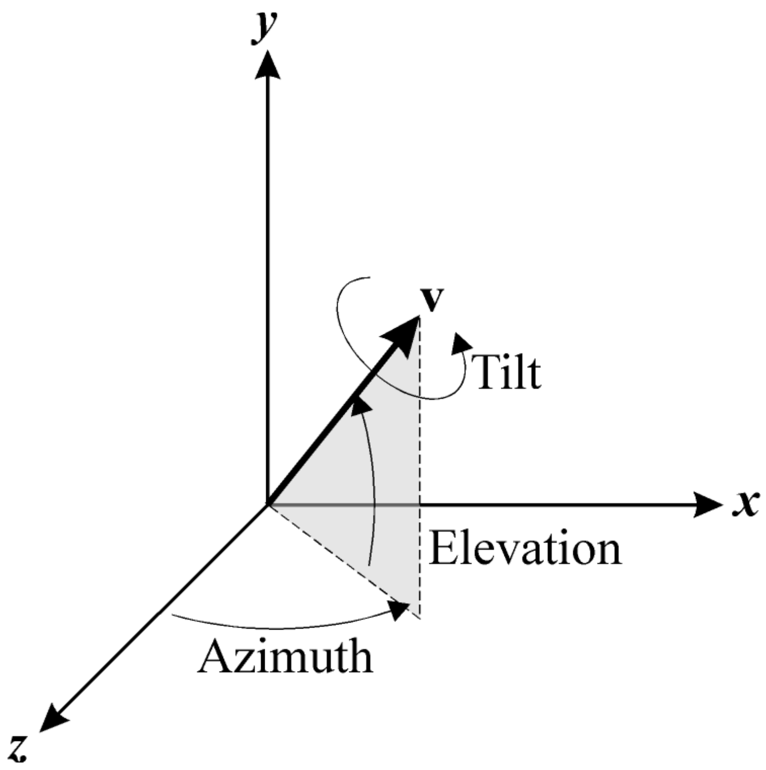
$$= \begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

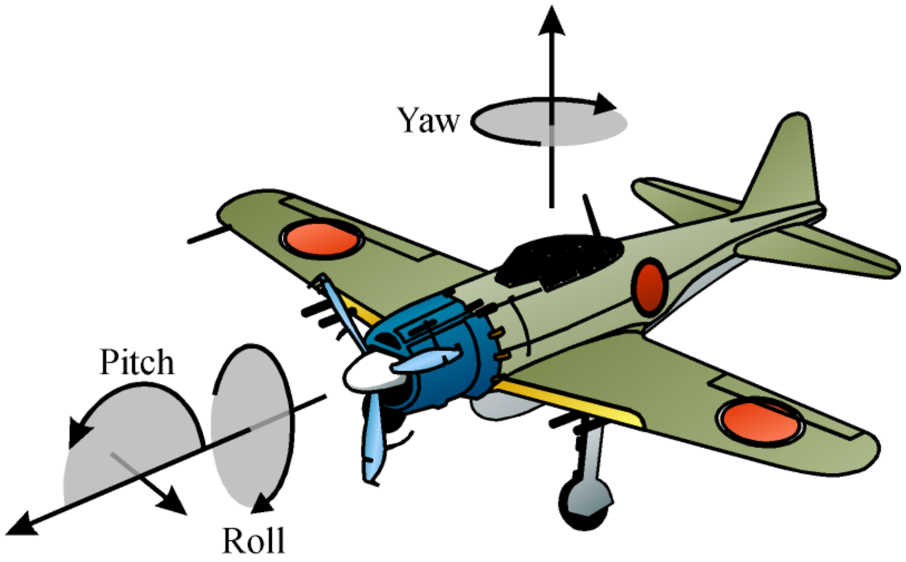
$$= \begin{bmatrix} \cos\theta & -\sin\theta & 0 & P_x - P_x\cos\theta + P_y\sin\theta \\ \sin\theta & \cos\theta & 0 & P_y - P_x\sin\theta - P_y\cos\theta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Euler Angles

- *Euler angles* represent the angles of rotation about the co-ordinate axes required to achieve a given orientation (θ_x, θ_y, θ_z)
- The resulting matrix is: $\mathbf{M} = \mathbf{R}(\theta_x)\mathbf{R}(\theta_y)\mathbf{R}(\theta_z)$
- Any required rotation may be described (*though not uniquely*) as a composition of 3 rotations about the coordinate axes.
- Remember rotation *does not commute* $\Rightarrow$ order is important

Rotational DOF





Sometimes known as *roll*, *pitch* and *yaw*

Extra Reading

- **Chapter 3: Geometric Objects and Transformations**
- Interactive Computer Graphics: A Top Down Approach with OpenGL, 6th Edition (or other) Angel
- **Chapter 4: Math for 3D Graphics**
- OpenGL Superbible, 6th Edition
- **Elementary Linear Algebra**, Anton
- "Homogeneous Coordinates and Computer Graphics" by Tom Davis
- http://www.geometer.org/mathcircles/cghomogen.pdf