Transformations

CSU44052 Computer Graphics

Binh-Son Hua

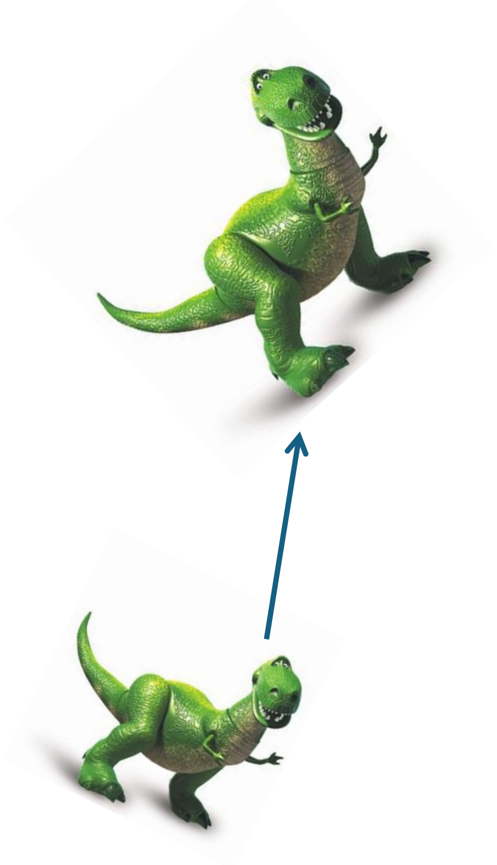
Slides adapted from the previous version by Carol O'Sullivan

Objectives

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



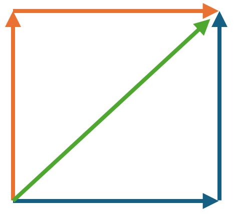
Geometric Transformations

- Many geometric transformations are *linear*.
- Function *f* is linear iff: $f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$
- Implications:
  - 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′*

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

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

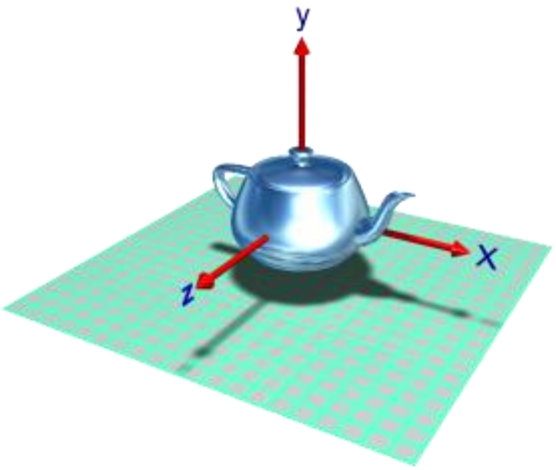
- A translation is defined by specifying the displacements a, b, and c

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

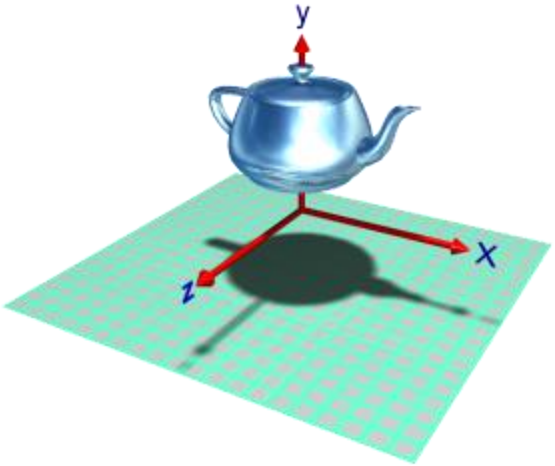
Translation

- Translation *only applies to points*, we never translate vectors.

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



translate along ***y***



Scaling

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

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

$$x' = s_x x$$

$$y' = s_y y$$

$$z' = s_z z$$

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)

- We can represent scaling operation using a matrix:

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

Recap on Matrices

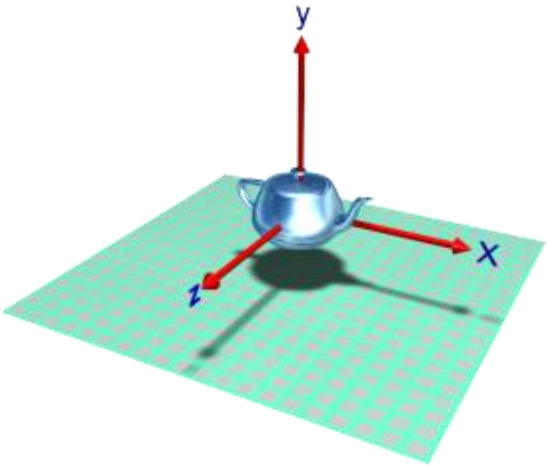
- Matrix addition
  - $\begin{bmatrix} a & c \\ b & d \end{bmatrix}$+$\begin{bmatrix} e & g \\ f & h \end{bmatrix}$ = $\begin{bmatrix} a+e & c+g \\ b+f & d+h \end{bmatrix}$
- Matrix multiplication
  - $\begin{bmatrix} a & c \\ b & d \end{bmatrix}x\begin{bmatrix} e & g \\ f & h \end{bmatrix}$ = $\begin{bmatrix} ae+cf & ag+ch \\ be+df & bg+dh \end{bmatrix}$
  - 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
  - Transpose A^T of a matrix A is one whose rows are switched with its columns

Non-Uniform Scaling

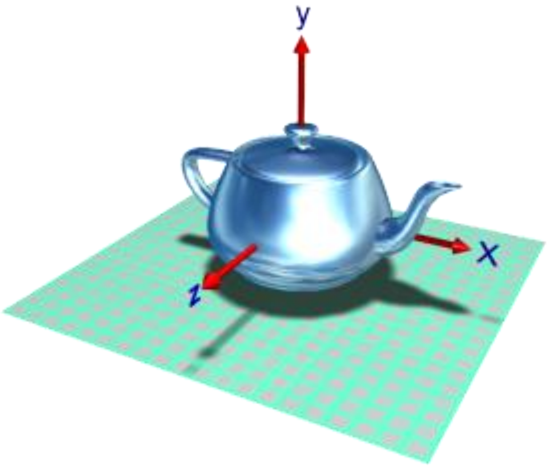
- Make an object twice as big in the x-direction
  - Multiply all x-coordinates by 2, leave y&z unchanged
- 3 times as large in the y-direction
  - Multiply all y-coordinates by 3, leave z&x unchanged

Scale

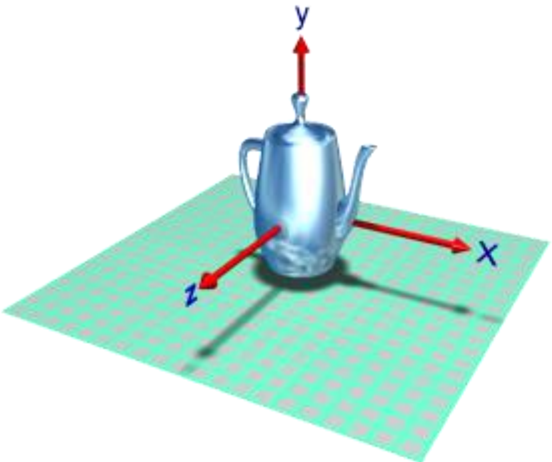
all vectors are scaled from the origin



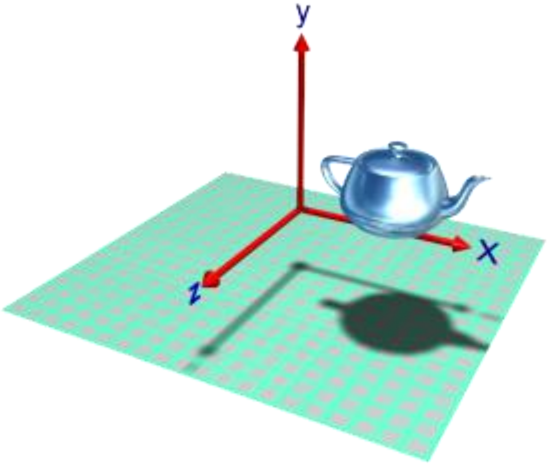
Original



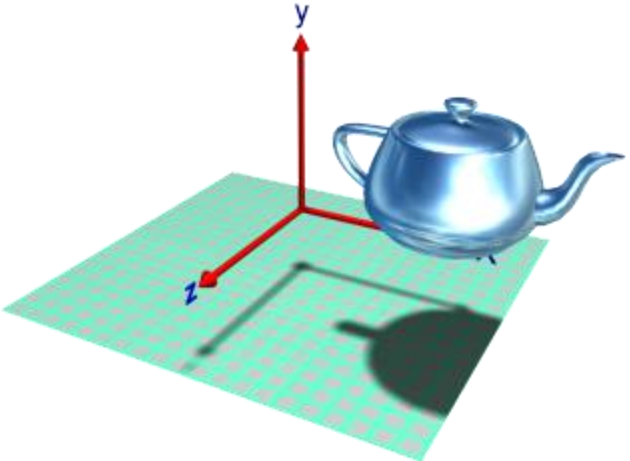
scale all axes



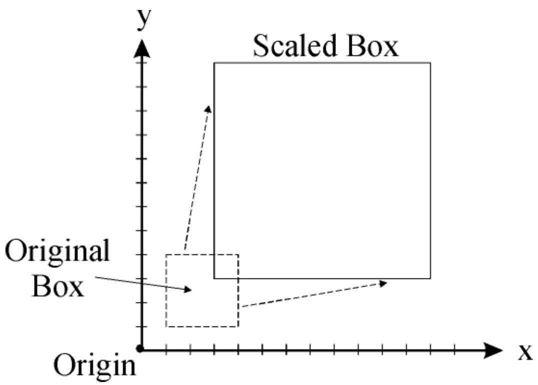
scale Y axis



offset from origin

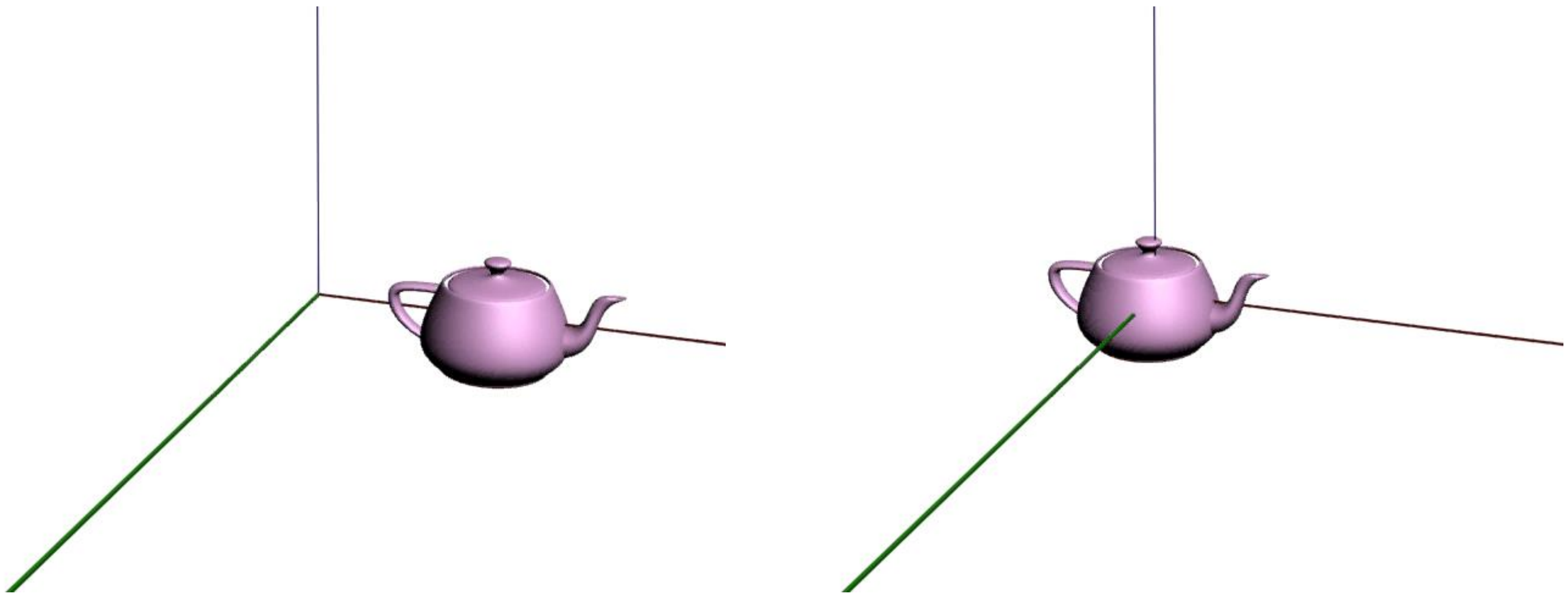


distance from origin also scales



Scaling an object

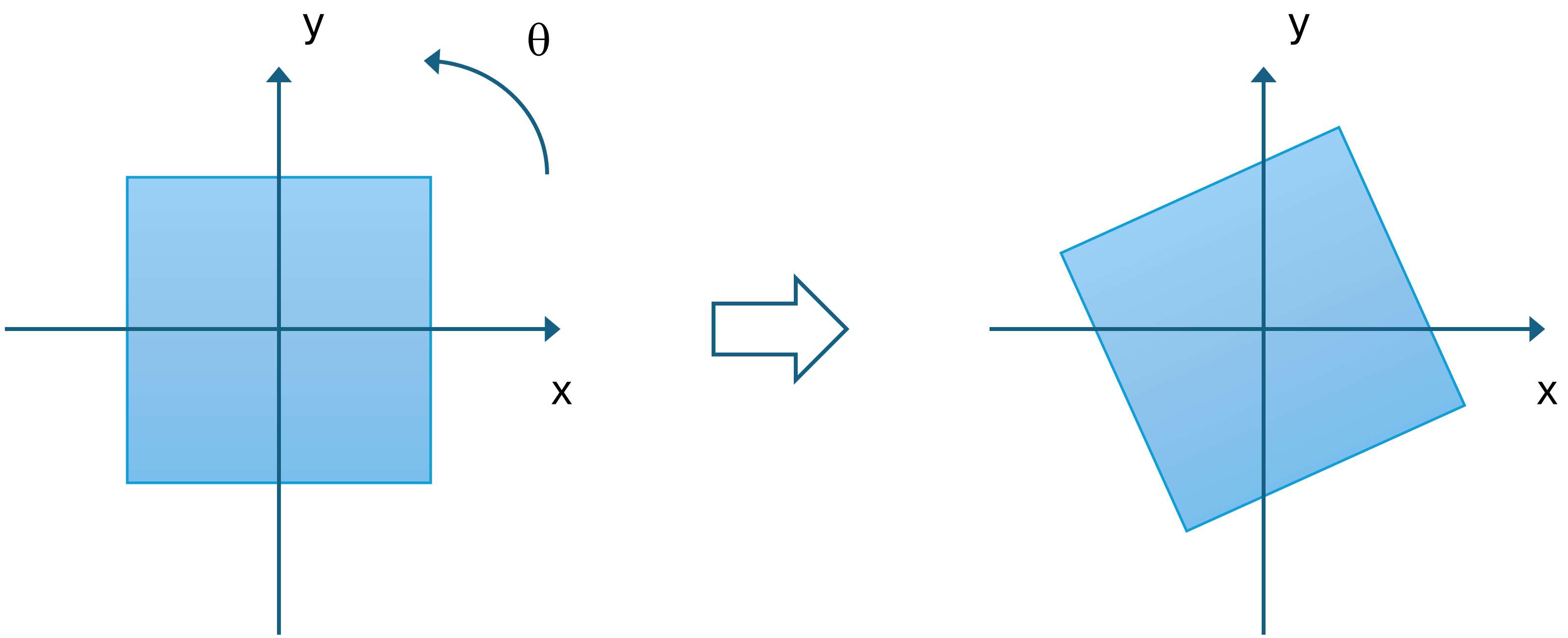
- 3 times smaller
- Multiply all our coordinates by 1/3
- We get a model that is 1/3 of the size
- However
  - If original coordinates described a car 1 mile from the origin
  - Miniature car would only be 1/3 mile from the origin
- Solution – translation to origin, and then scale, then translate back



2D 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 Rotation

- Consider the 2D rotation in the x-y plane about the origin by an angle ϴ in counter-clockwise direction
  - Same as rotation about the z-axis
- Let’s generalize 2D rotation to 3D

Rotation

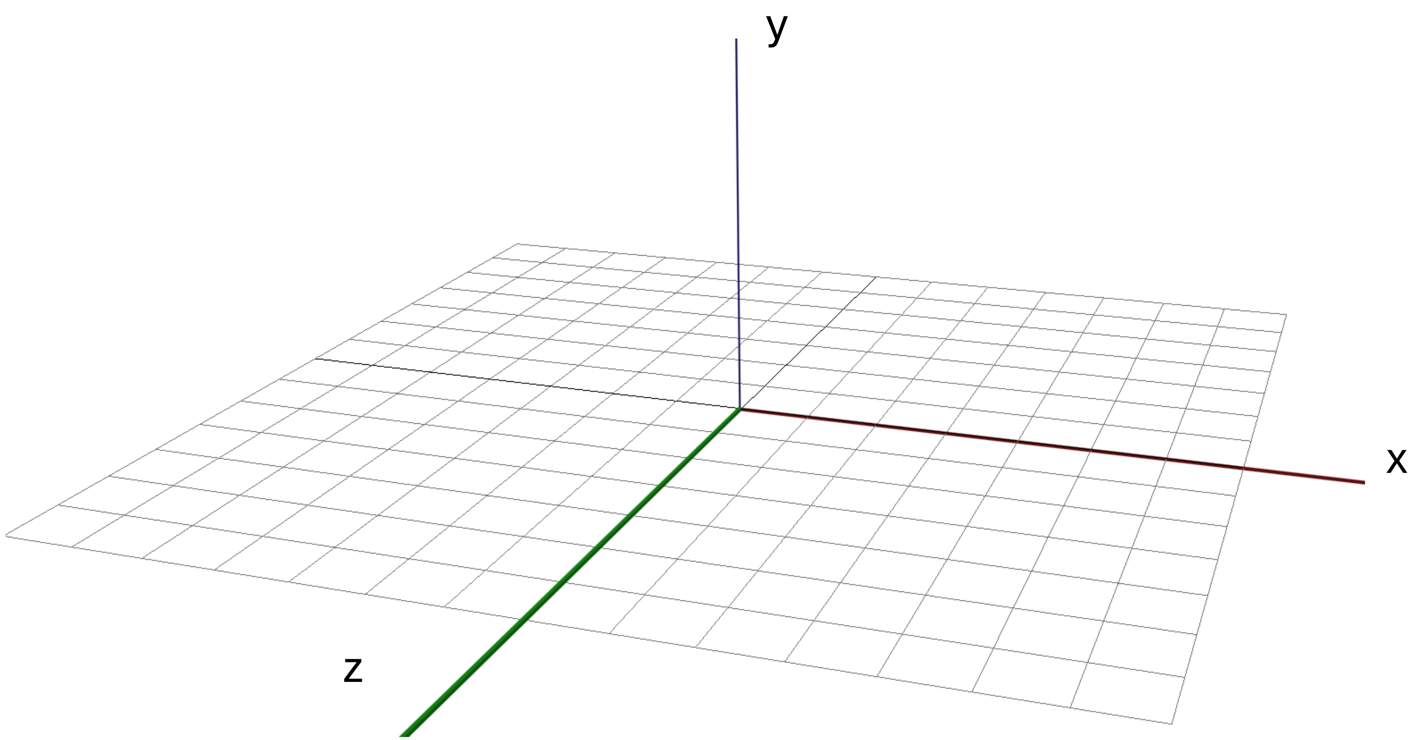
Definition (Rotation)
A rotation turns about a point (a,b) through an angle θ

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

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

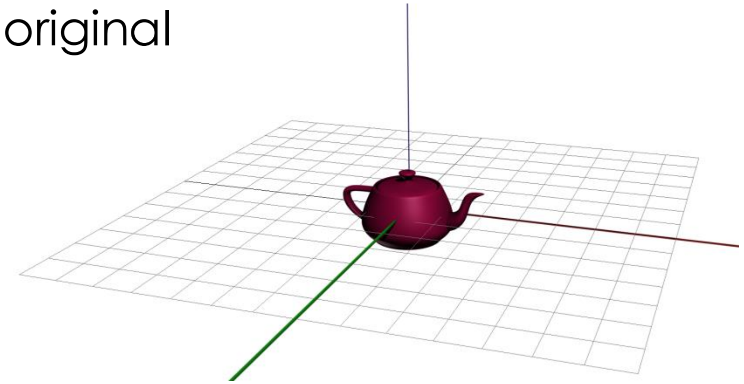
Rotation - idea

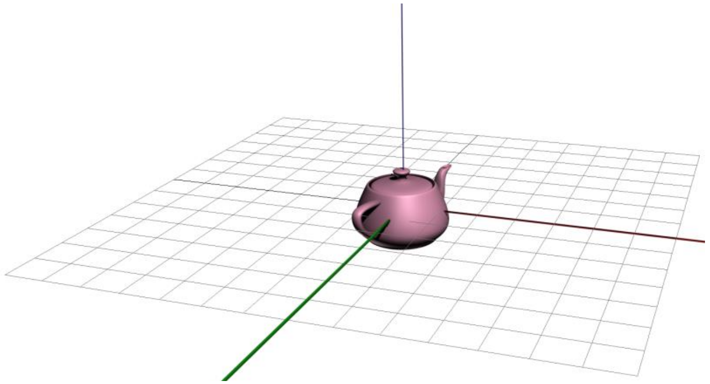
- Visualise rotation about an axis:
  - Put your eye on that axis in the positive direction and look towards the origin
  - Then, a positive rotation corresponds to a counter-clockwise rotation

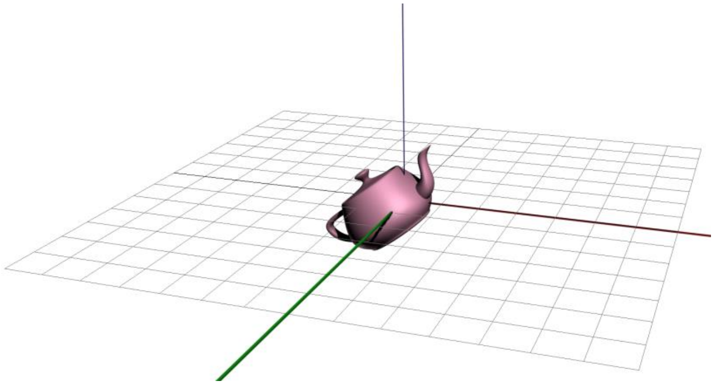


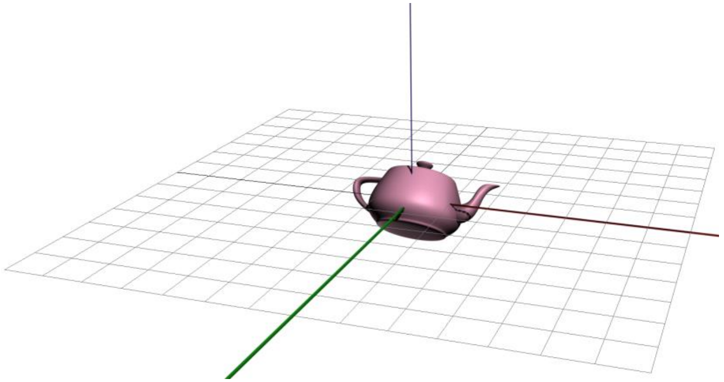
Which Axis?

original



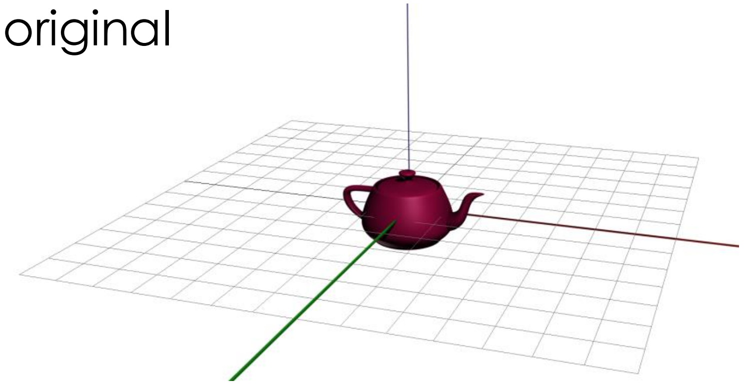




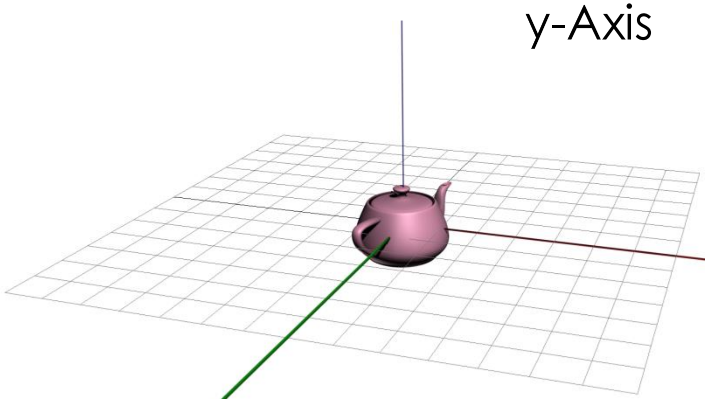


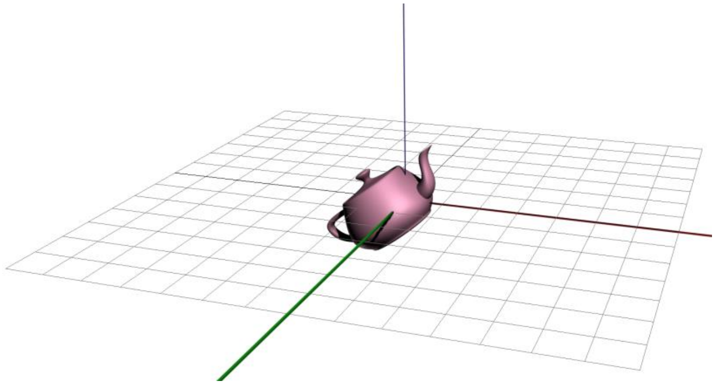
Which Axis?

original

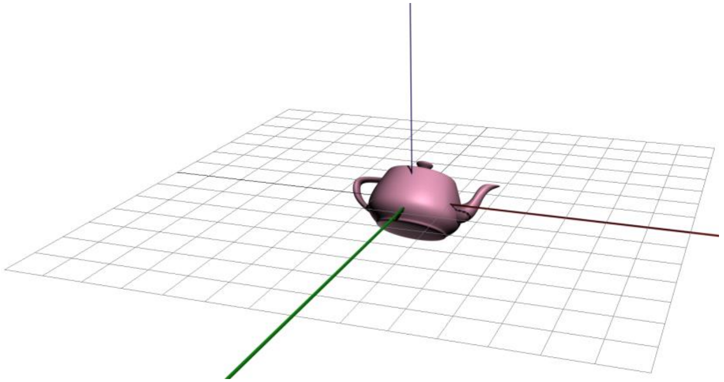


y-Axis





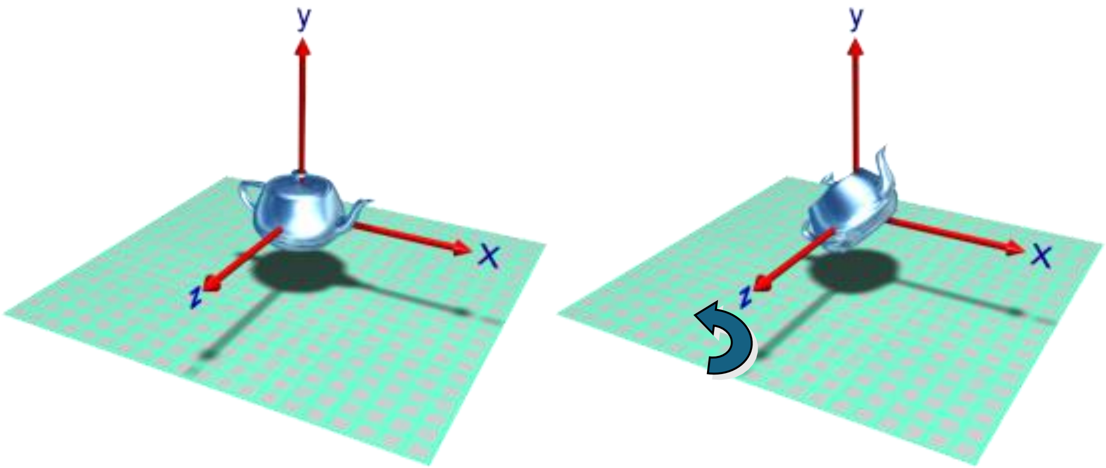
z-axis



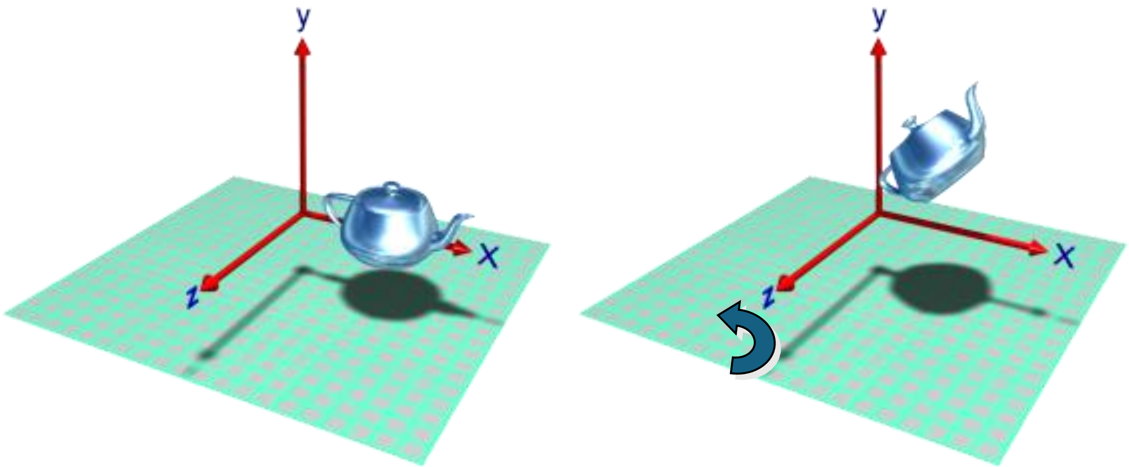
(-) x-axis

Rotation

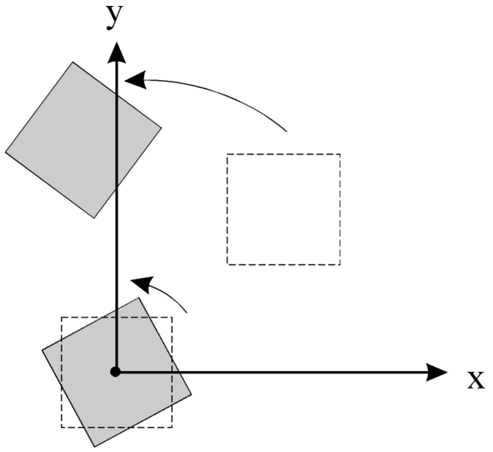
Rotations are *anti-clockwise* about the *origin*



rotation of 45° about the Z axis



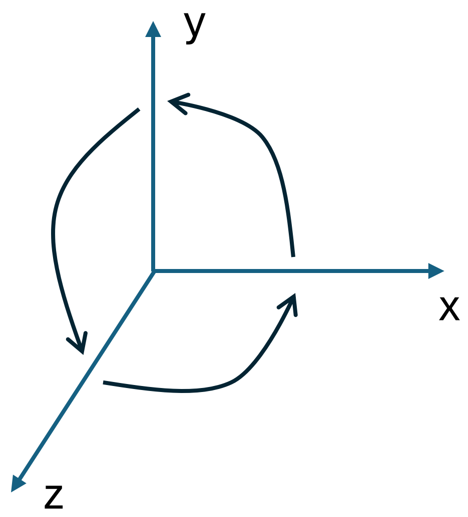
offset from origin rotation



Rotation Matrices

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \qquad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Note: difference for rotation about y, due to RHS



Rotation

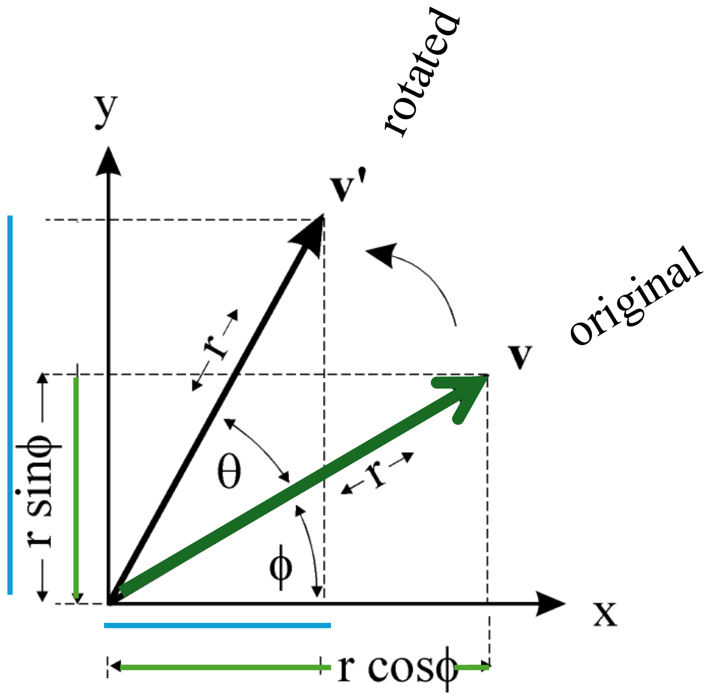
- Rotation in the clockwise direction is the **inverse** of rotation in the counter-clockwise direction and vice versa

$$\begin{aligned}\cos(-\theta) &= \cos\theta \\ \sin(-\theta) &= -\sin\theta\end{aligned} \Rightarrow \mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta) = \mathbf{R}^{T}(\theta)$$

- Linear algebra: $\mathbf{M}^{-1} = \mathbf{M}^{T}$ then **M** is *orthonormal*.
  All orthonormal matrices are rotations about the origin.

Rotation about the z-axis

$$\mathbf{v} = \begin{bmatrix} r\cos\phi \\ r\sin\phi \end{bmatrix} \quad \mathbf{v}' = \Bigg[$$

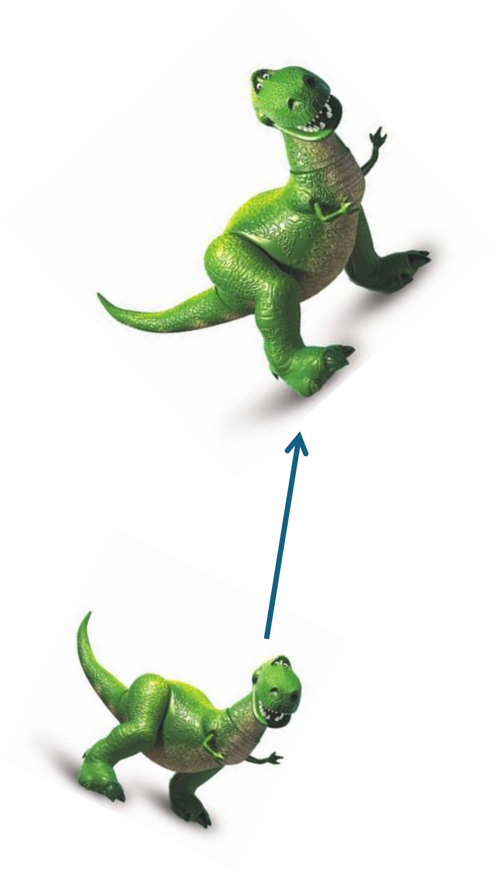


$$\text{expand } (\phi+\theta) \Rightarrow \begin{cases} x' = r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ y' = r\cos\phi\sin\theta + r\sin\phi\cos\theta \end{cases}$$

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

Combining Rotation, Scaling, Translation

- 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: $\mathbf{M} = \mathbf{RS}$



- How can we handle **translation** using matrices?

Homogeneous Coordinates

- Basis of the homogeneous coordinate 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 Coordinates

- 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

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

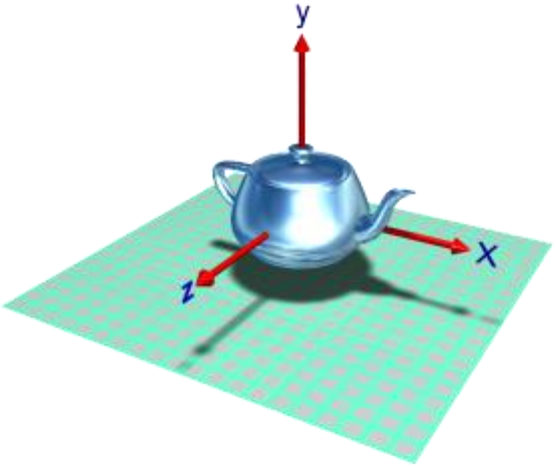
Homogenous Coordinates

- 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

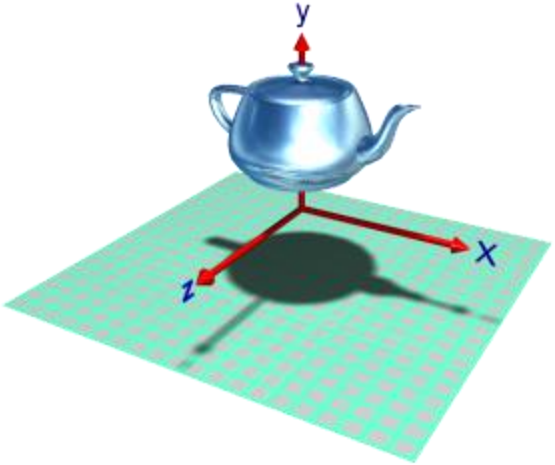
Translation in Homogeneous Coordinates

- Translation *only applies to points*, we never translate vectors.
- Remember: 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***



Scale in Homogeneous Coordinates

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

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

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

Rotation in Homogeneous Coordinates

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

Transformation Composition

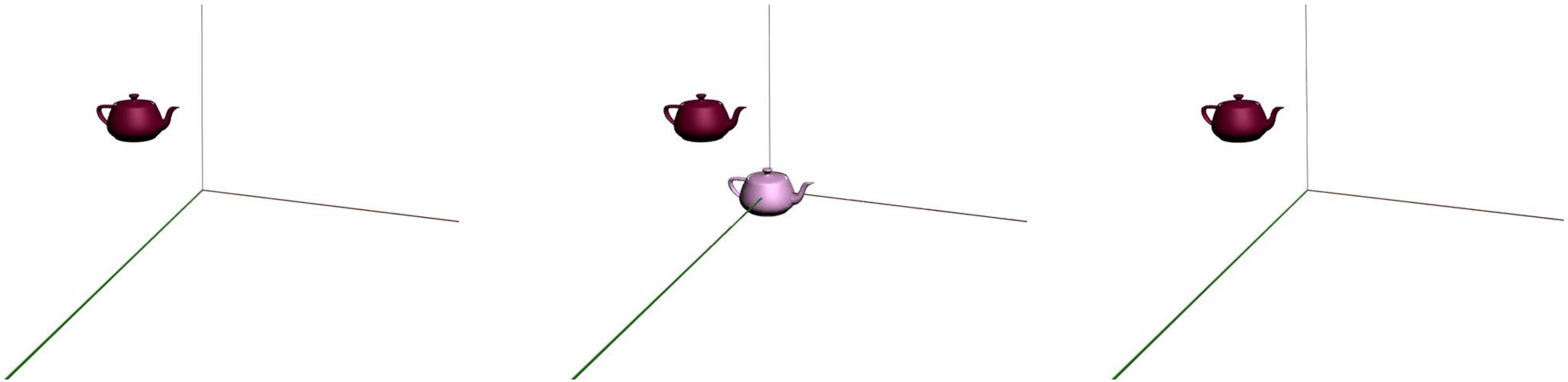
- More complex transformations can be created by *concatenating* or *composing* individual transformations together.

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

- Matrix multiplication is *non-commutative* $\Rightarrow$ **order is vital**
- $\mathbf{T} \circ \mathbf{R} \circ \mathbf{S} \circ \mathbf{T}$ *means* $\mathbf{T}$ $after$ $\mathbf{R}$ $after$ $\mathbf{S}$ $after$ $\mathbf{T}$, *i.e., order is* ***reversed***

Rotation about a point

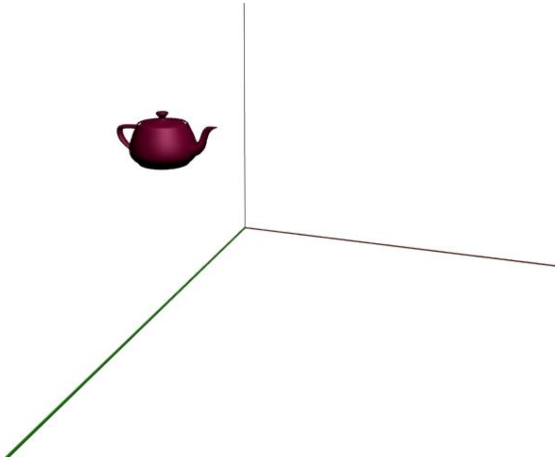
- What if rotation is not about the origin?
  - Translate the centre of rotation to the origin,
  - Perform the rotation
  - Translate back

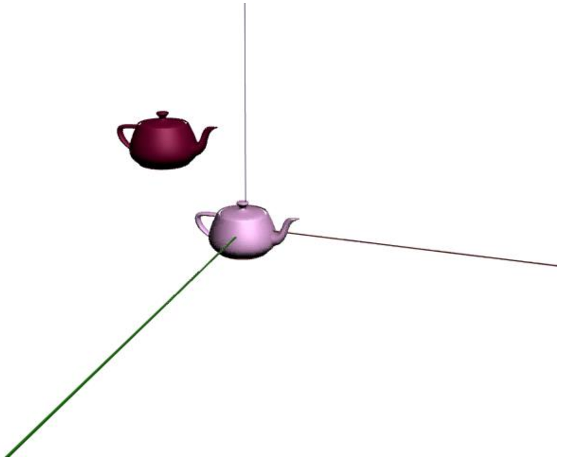


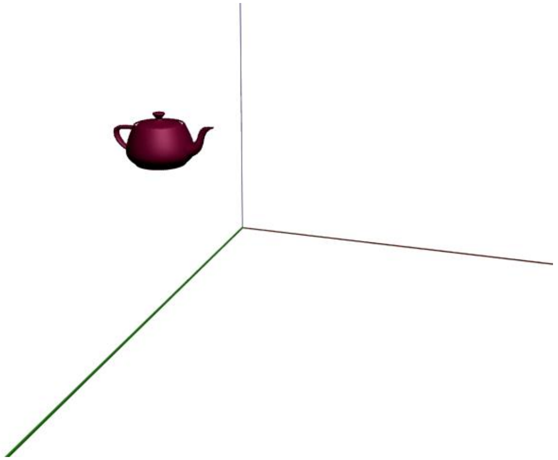
Rotation about a point

- We can create an affine transformation representing rotation about a point P_R: *translate to origin, rotate about origin, translate back to original location*

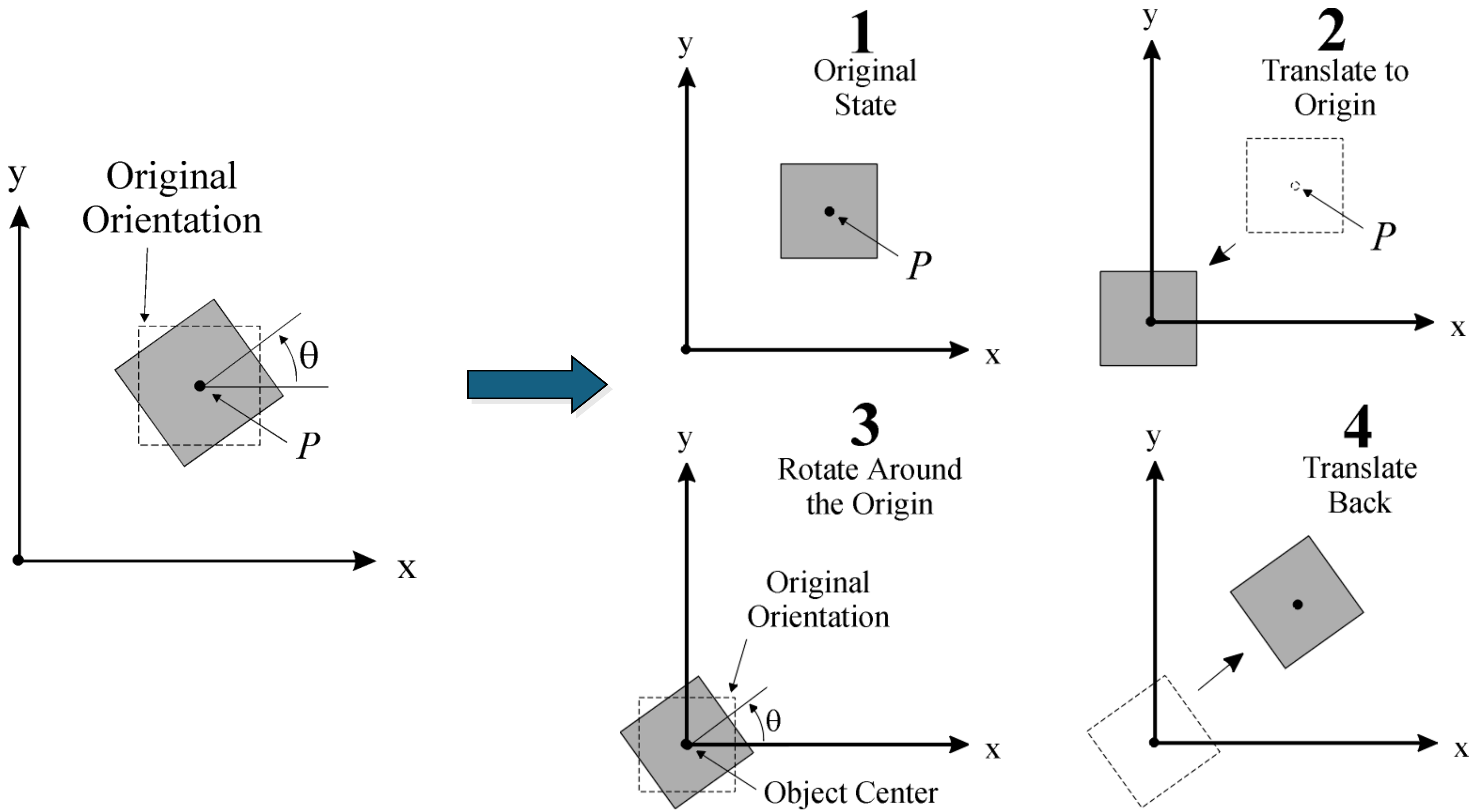
$$\mathbf{M} = \mathbf{T}(P_R)\mathbf{R}(\theta)\mathbf{T}(-P_R)$$







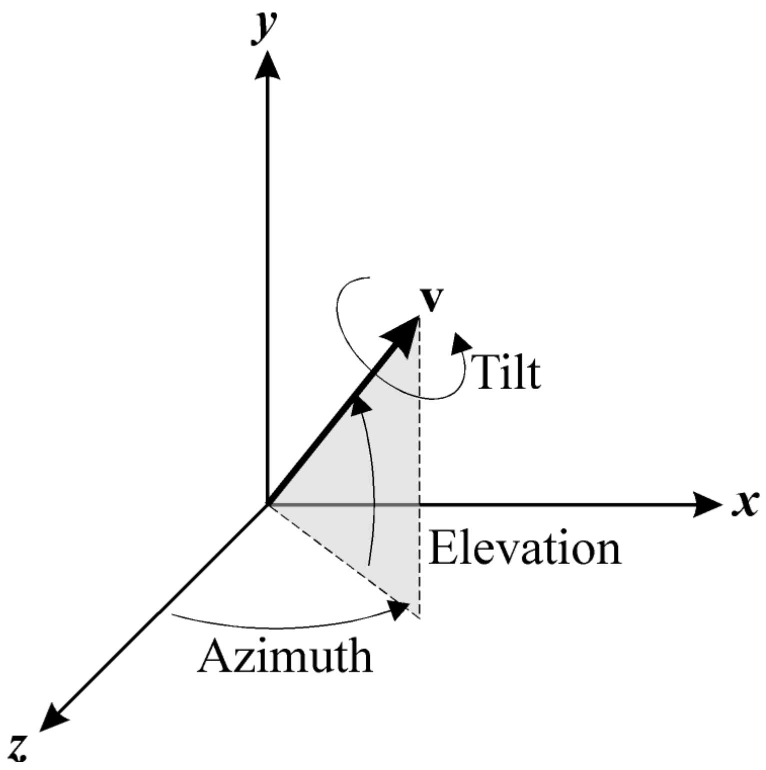
Transformation Composition

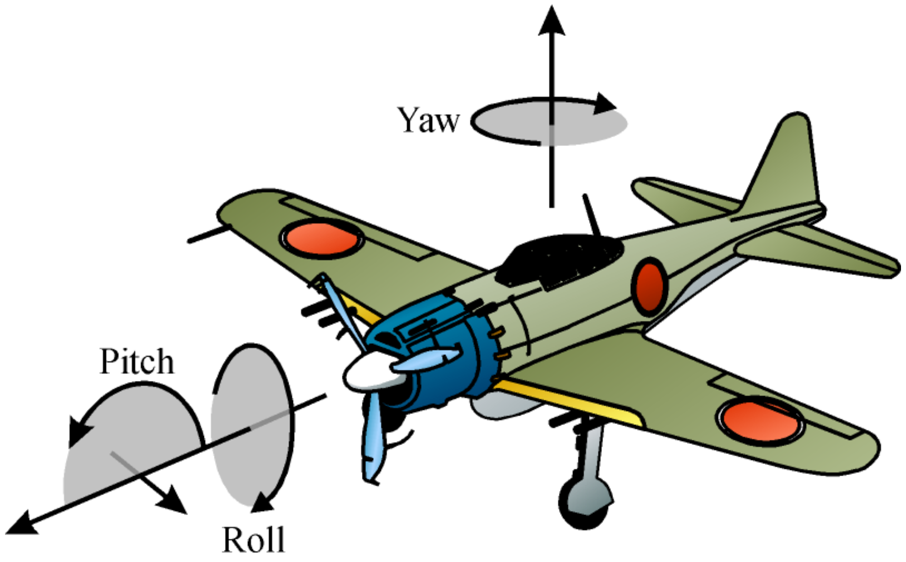


Rotation using Euler Angles

- *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



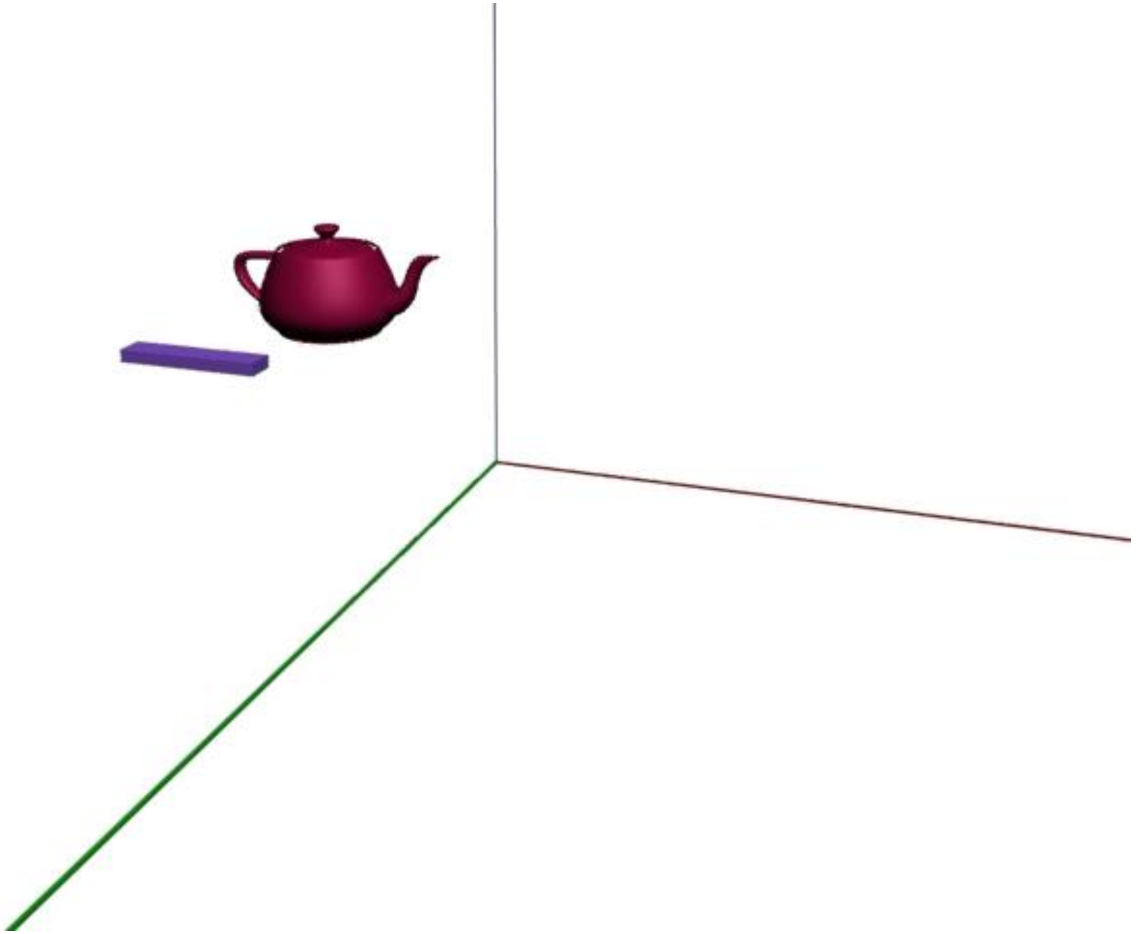


Sometimes known as *roll*, *pitch* and *yaw*

General Rotation

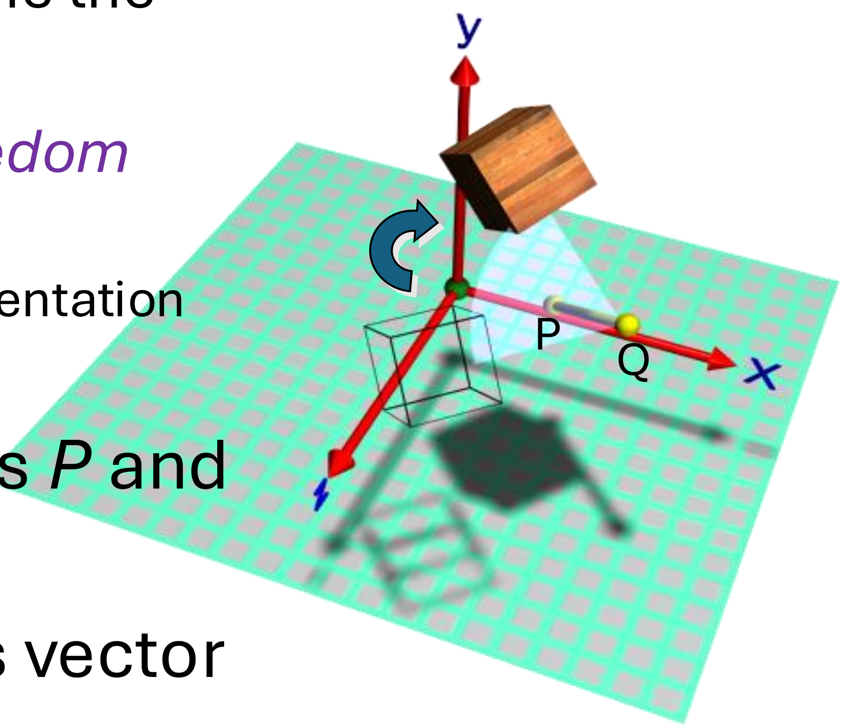
- What if you want to rotate about an axis that does not happen to be one of the 3 principle axes?
  - Can do this using operations we already have
- Strategy:
  - Do one or two rotations about the principal axes to get the axis we want aligned with the z-axis
  - Then, rotate about the z-axis
  - Undo the rotations we did to align your axis with the z-axis

Rotation about an arbitrary axis



Rotation about an arbitrary axis

- A frequent requirement is to determine the matrix for rotation about a given axis.
- Such rotations have 3 *degrees of freedom* (DOF):
  - 2 for spherical angles specifying axis orientation
  - 1 for twist about the rotation axis
- Assume axis is defined by points *P* and *Q*
- Pivot point is *P* and rotation axis vector is:



$$\mathbf{v} = \frac{P - Q}{\left| P - Q \right|}$$

v is a unit vector

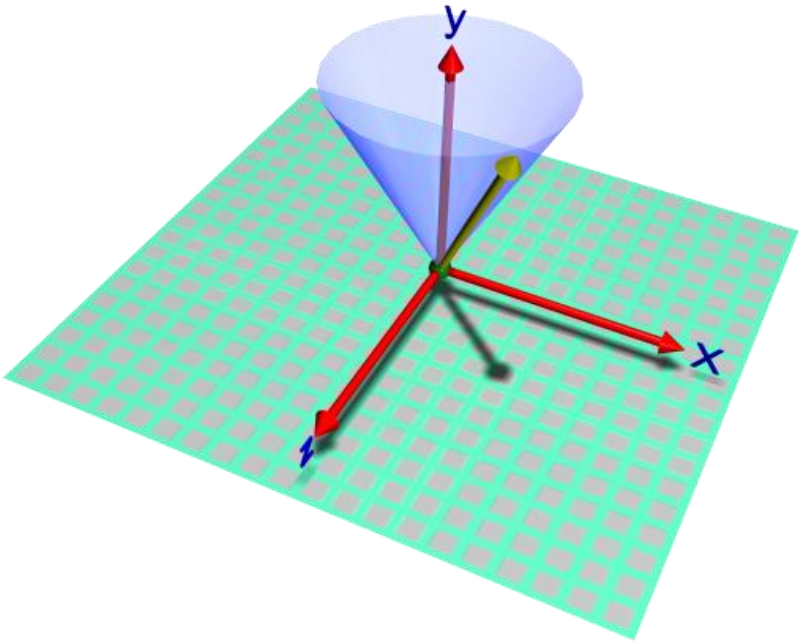
Rotation about an arbitrary axis

1. Translate the pivot point of the axis to the origin $\Rightarrow$ **T**$(-P)$
2. Rotate the **axis** and **object** so that the **axis** lines up with **z** say $\Rightarrow$ **R**(θ_y)**R**(θ_x)
3. Rotate about **z** by the required angle $\theta \Rightarrow$ **R**(θ)
4. Undo the first 2 rotations to bring us back to the original orientation $\Rightarrow$ **R**$(-\theta_x)$**R**$(-\theta_y)$
5. Translate back to the original position $\Rightarrow$ **T**(P)
6. The final rotation matrix is:

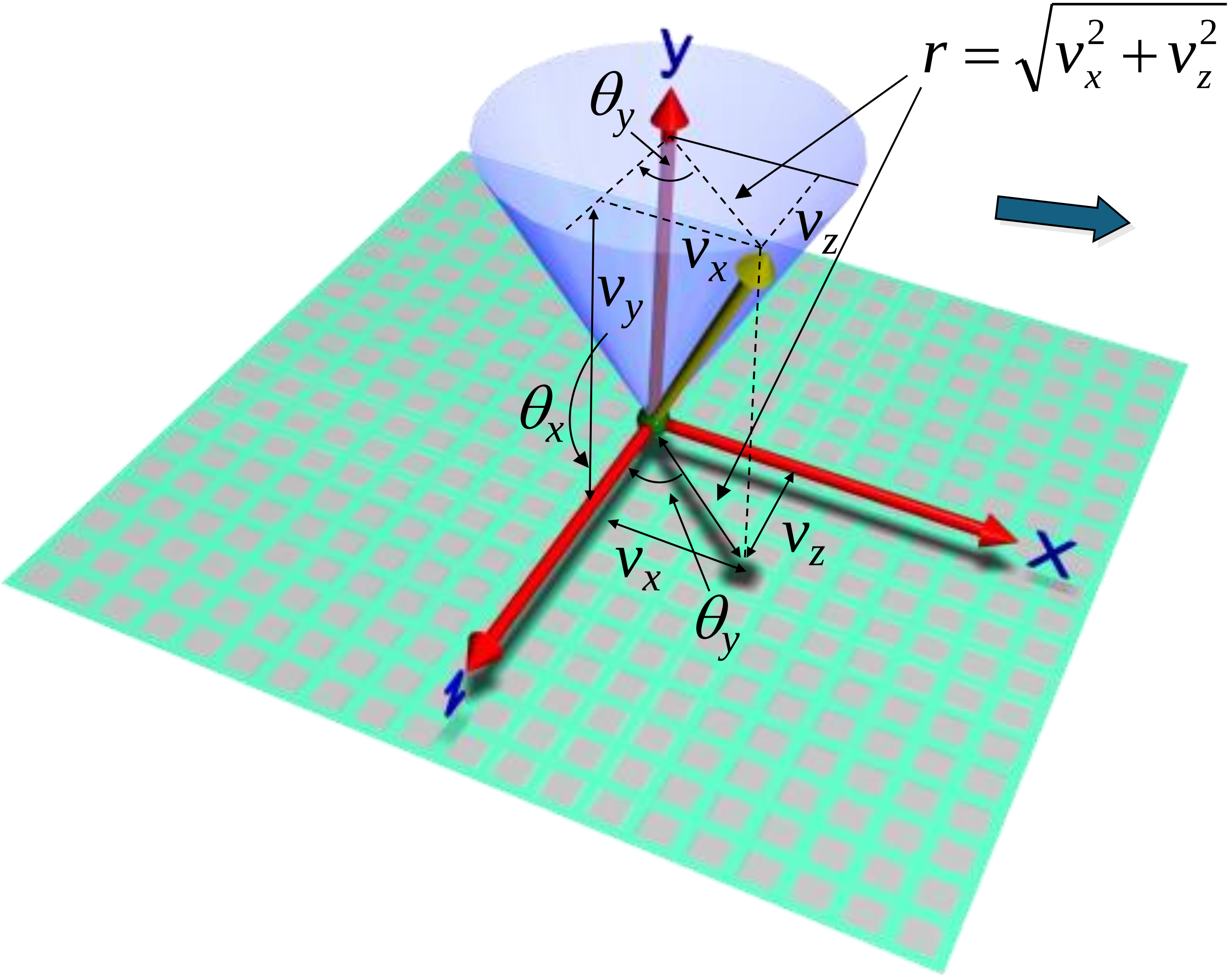
$$\mathbf{M} = \mathbf{T}(P)\mathbf{R}(-\theta_y)\mathbf{R}(-\theta_x)\mathbf{R}(\theta)\mathbf{R}(\theta_x)\mathbf{R}(\theta_y)\mathbf{T}(-P)$$

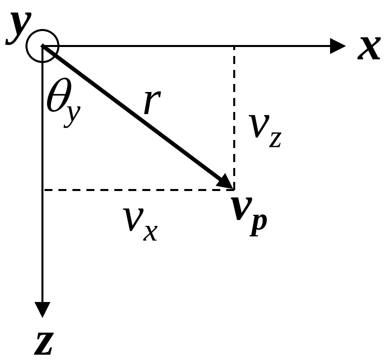
Rotation about an axis

- We need the Euler angles θ_x and θ_y which will orient the rotation axis along the **z** axis.
- We determine these using simple trigonometry.



Aligning axis with *z*

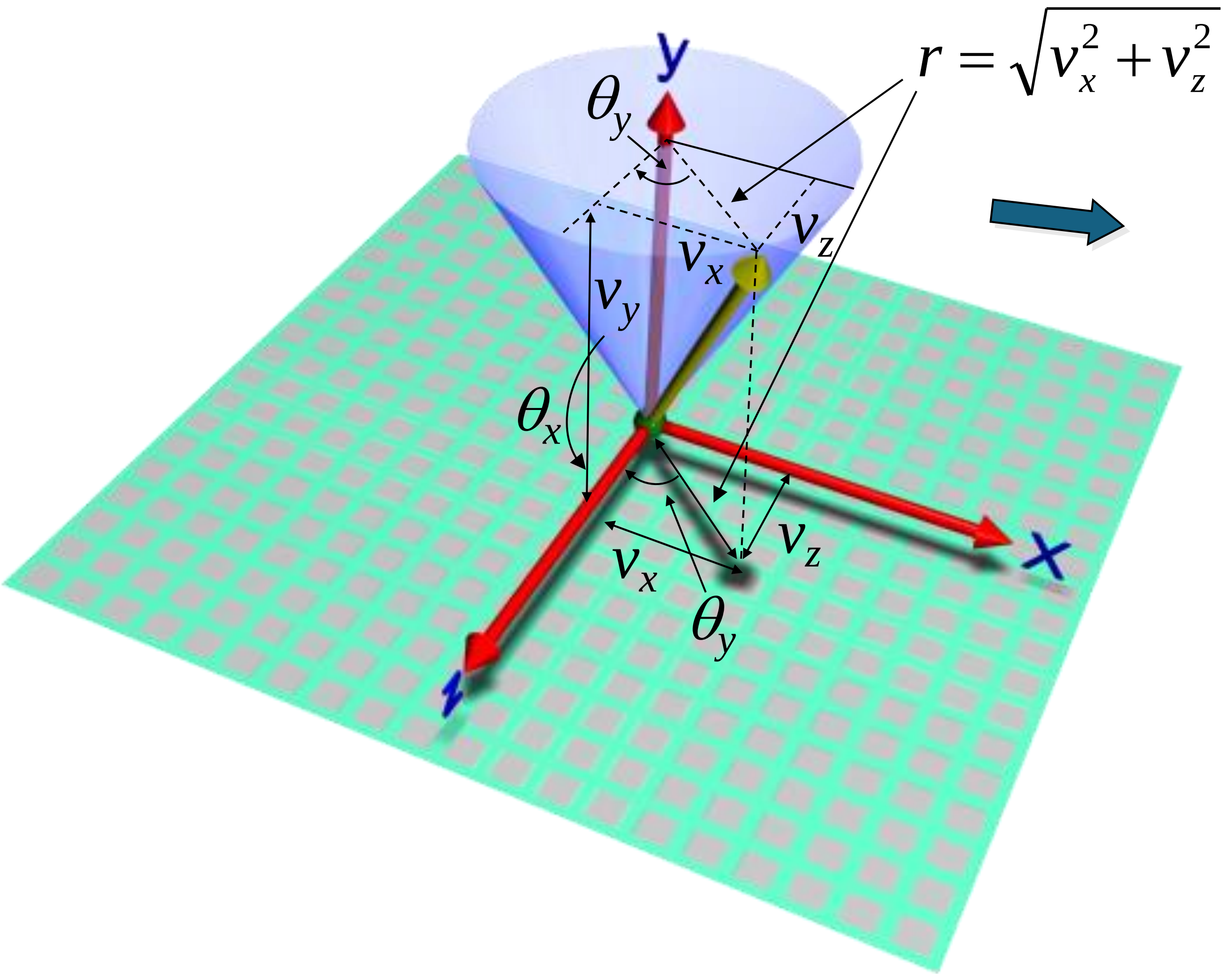


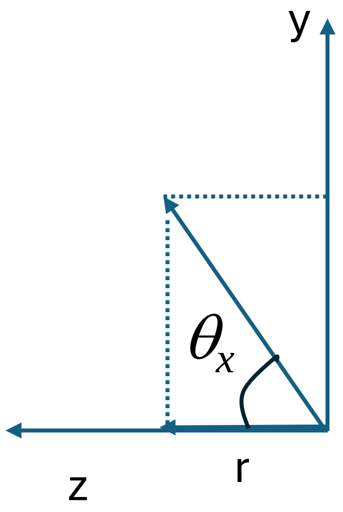


$$\cos\theta_y = \frac{v_z}{r}$$

$$\sin\theta_y = \frac{v_x}{r}$$

Aligning axis with *z*





$$\cos\theta_x = r$$

$$\sin\theta_x = v_y$$

Aligning axis with z

- Note that as shown the rotation about the **x** axis is anti-clockwise but the **y** axis rotation is *clockwise*.
- Therefore, the angle for **y** axis rotation needs to be negated, which is -$\theta_y \Rightarrow$

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

$$\mathbf{R}(\theta_x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & r & -v_y & 0 \\ 0 & v_y & r & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \mathbf{R}(-\theta_x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & r & v_y & 0 \\ 0 & -v_y & r & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}(\theta_y) = \begin{bmatrix} v_z/r & 0 & v_x/r & 0 \\ 0 & 1 & 0 & 0 \\ -v_x/r & 0 & v_z/r & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \mathbf{R}(-\theta_y) = \begin{bmatrix} v_z/r & 0 & -v_x/r & 0 \\ 0 & 1 & 0 & 0 \\ v_x/r & 0 & v_z/r & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Further Reading

- Homogeneous Coordinates and Computer Graphics by Tom Davis
  http://www.geometer.org/mathcircles/cghomogen.pdf
- Chapter 3: Geometric Objects and Transformations
  Interactive Computer Graphics: A Top Down Approach with OpenGL, 6th Edition, Angel and Shreiner
- Elementary Linear Algebra, Howard Anton