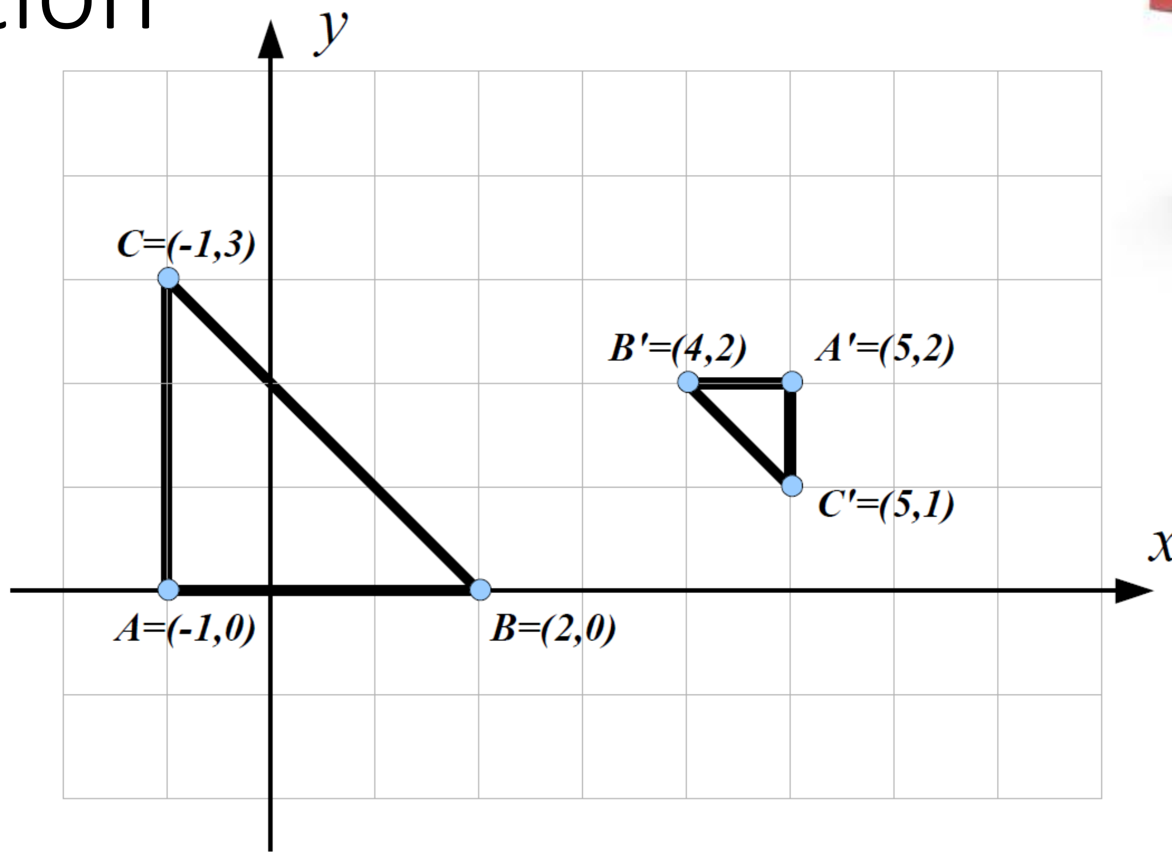
3D Transformations

COMP557

Paul Kry

Question

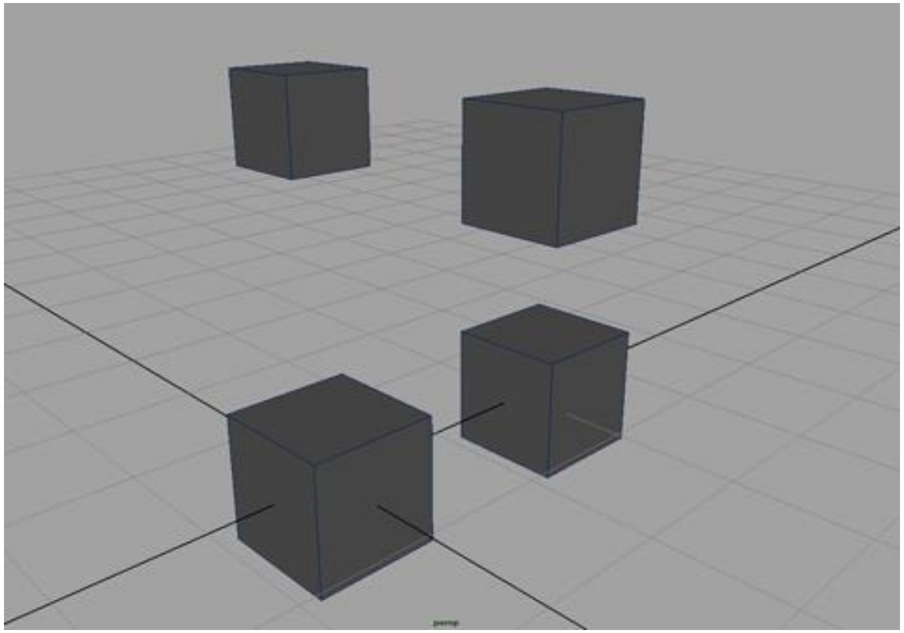


Give a homogeneous transformation matrix, or product of matrices, that will transform triangle ABC to triangle A’B’C’.

Based on slides by Steve Marschner

Translation

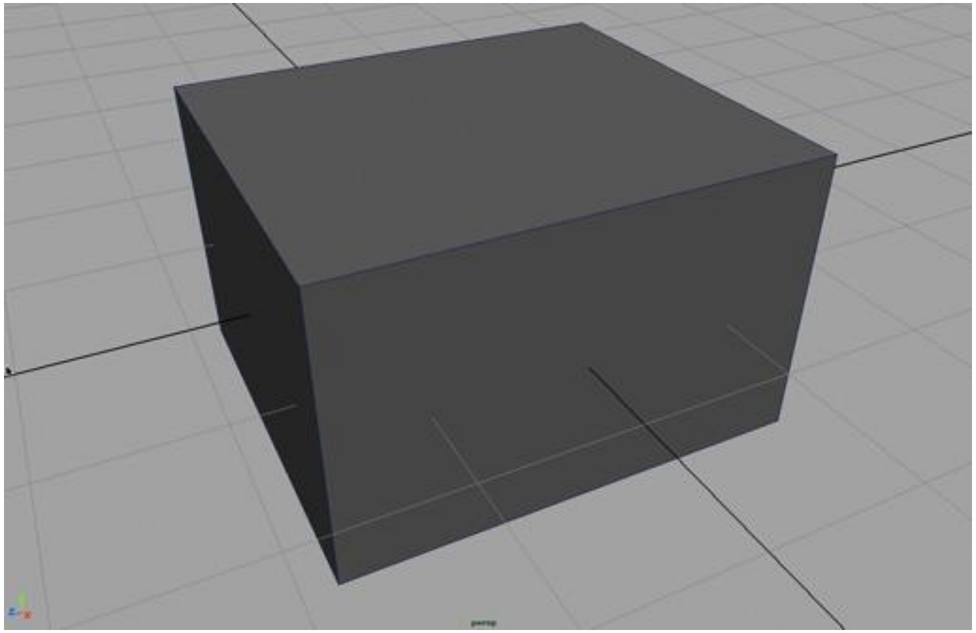
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Based on slides by Steve Marschner

Scaling

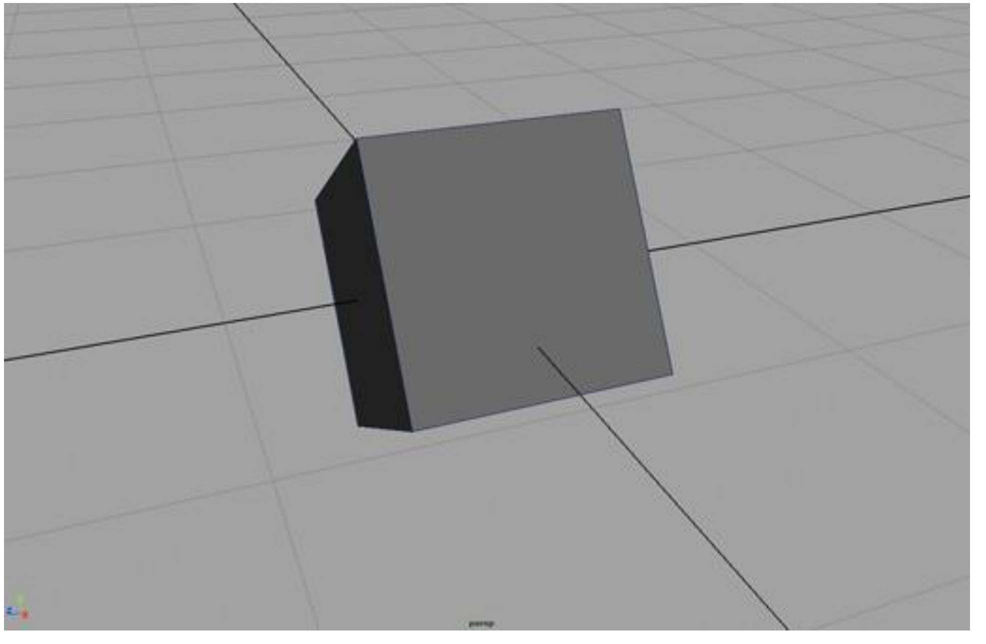
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Based on slides by Steve Marschner

Rotation about -*z* axis

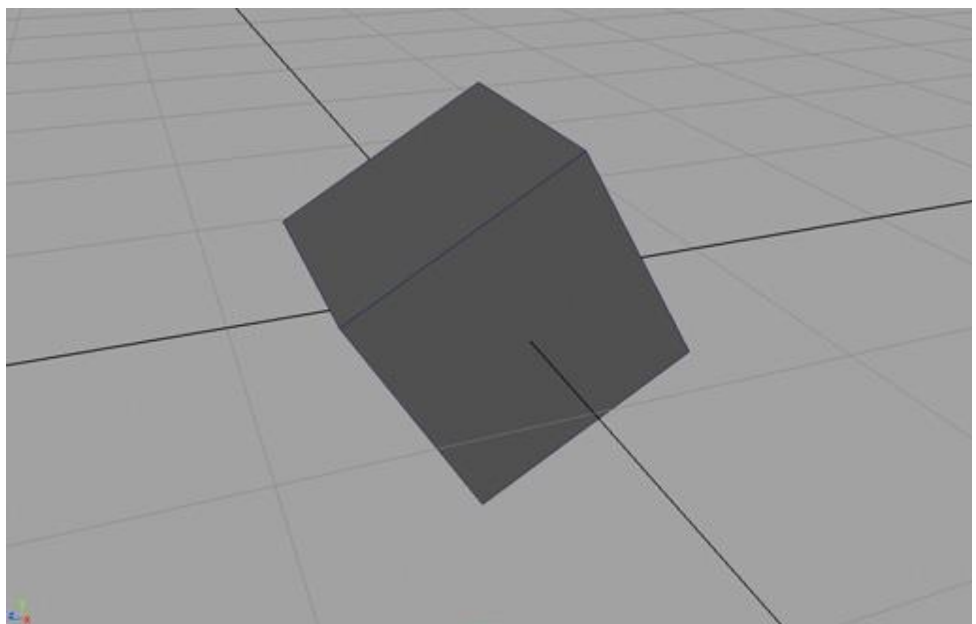
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Based on slides by Steve Marschner

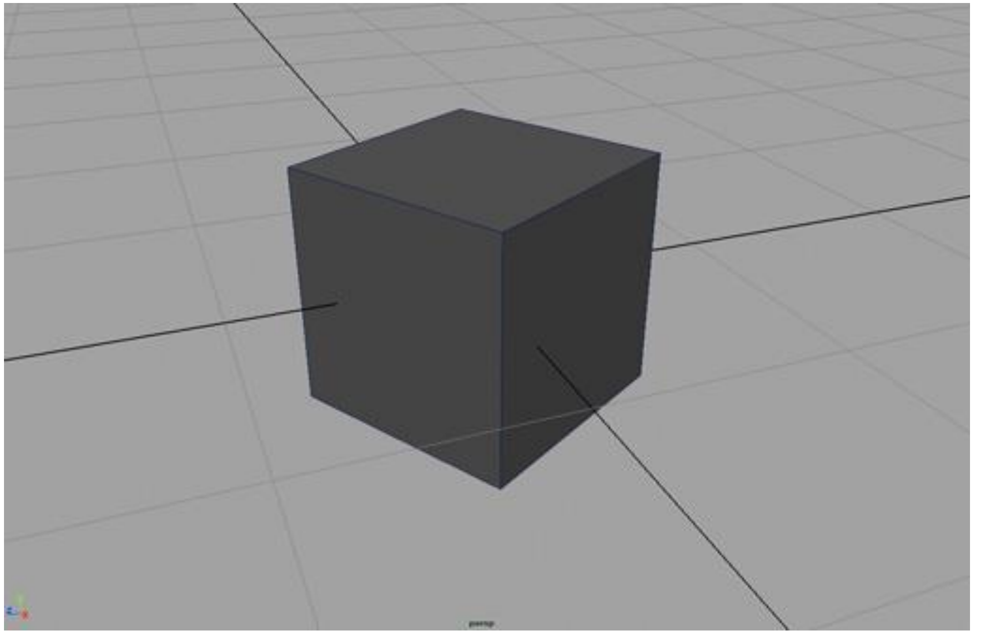
Rotation about -*x* axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



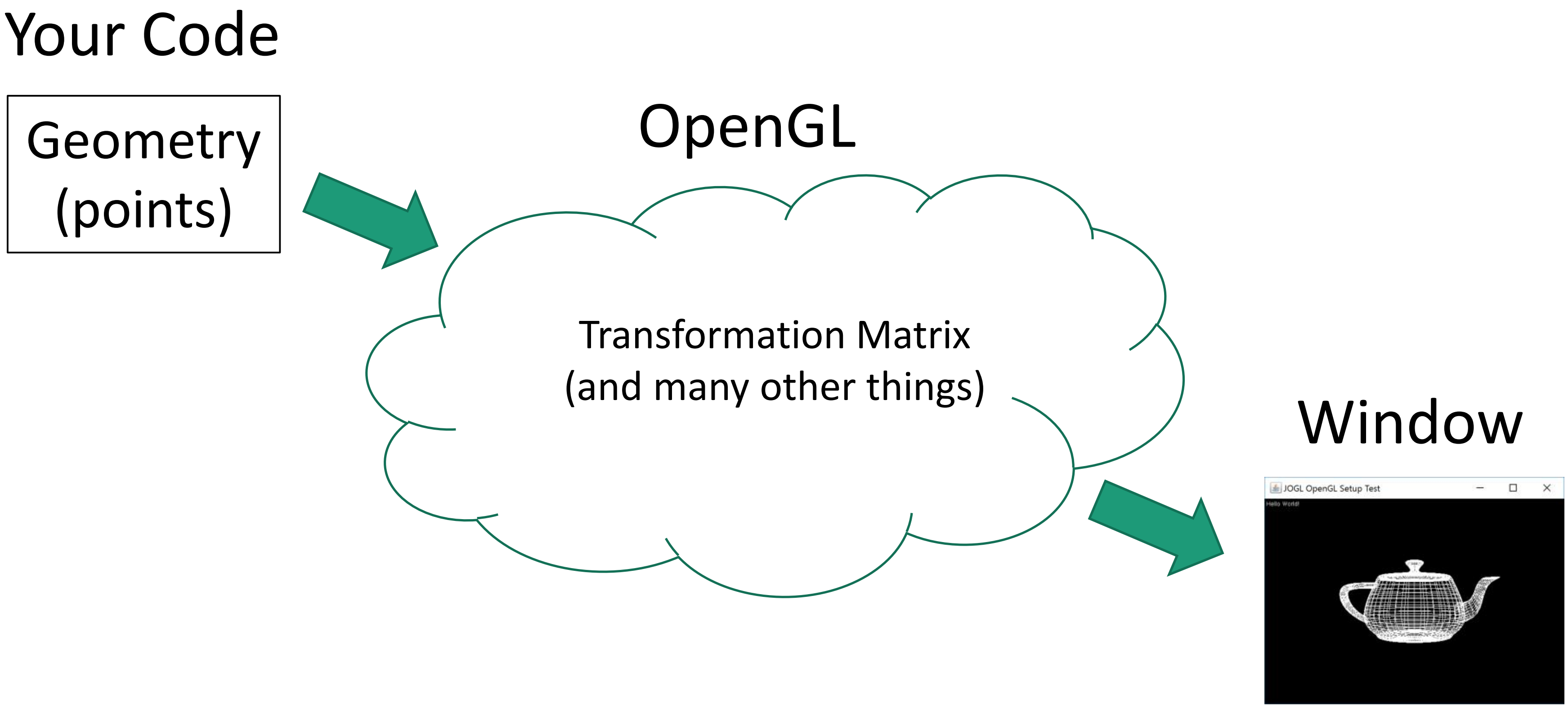
Rotation about -*y* axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Based on slides by Steve Marschner

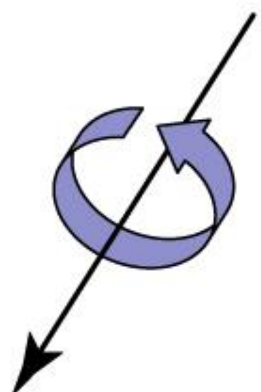
Transformations in OpenGL



General rotations

- A rotation in 2D is around a point
- A rotation in 3D is around an axis
  - so 3D rotation is w.r.t a line, not just a point
  - there are many more 3D rotations than 2D
    - a 3D space around a given point, not just 1D

2D



3D

In 3D, one way to specify a Rotation is via a unit vector (the axis) and an angle. Convention: positive rotation is CCW when vector points points toward you.

Based on slides by Steve Marschner

What are rotations? More precisely…

3D Rotations are the set of linear transformations

$$\{R \in \mathbb{R}^{3\times 3} : R^T = R^{-1}, \det(R) = +1\}$$

which form a group with matrix multiplication as the binary group operation (i.e., operation that combines two elements)

- Closed under the group operation
- Exist identity element (the identity matrix)
- Exist an inverse for every group element
- Associative, A(BC) = (AB)C

This group is the “special orthogonal group” SO(3). Each different possible 3D *orientation* is specified only once in this group.

Based on slides by Steve Marschner

Specifying rotations

- In 2D, a rotation just has an angle
  - How many Degrees Of Freedom, i.e., DOFs?

    $R^T = R^{-1}$ det(R) = +1 does not perhaps give much intuition, but

    knowing that an angle about an axis direction lets us identify 3 DOFs
- Can specify a rotation matrix with Euler angles
  - stack up three coordinate axis rotations, but what order??
  - Problem of gimbal lock !! (more on this shortly)

Euler's Theorem

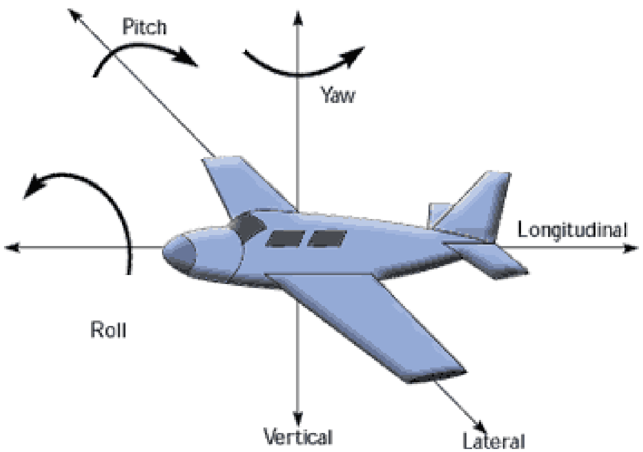
- Any two independent orthonormal coordinate frames can be related by a sequence of rotations (not more than three) about coordinate axes, where no two successive rotations may be about the same axis.
  - Have 12 possible sequences!

XYZ	**XZY**	**XYX**	**XZX**	**YXZ**	**YZX**
YXY	**YZY**	**ZXY**	**ZYX**	**ZXZ**	**ZYZ**

- Given some Euler angles, not knowing the order or if they map frame A to B or vice versa, then there is 24 possibilities!

Euler Angles

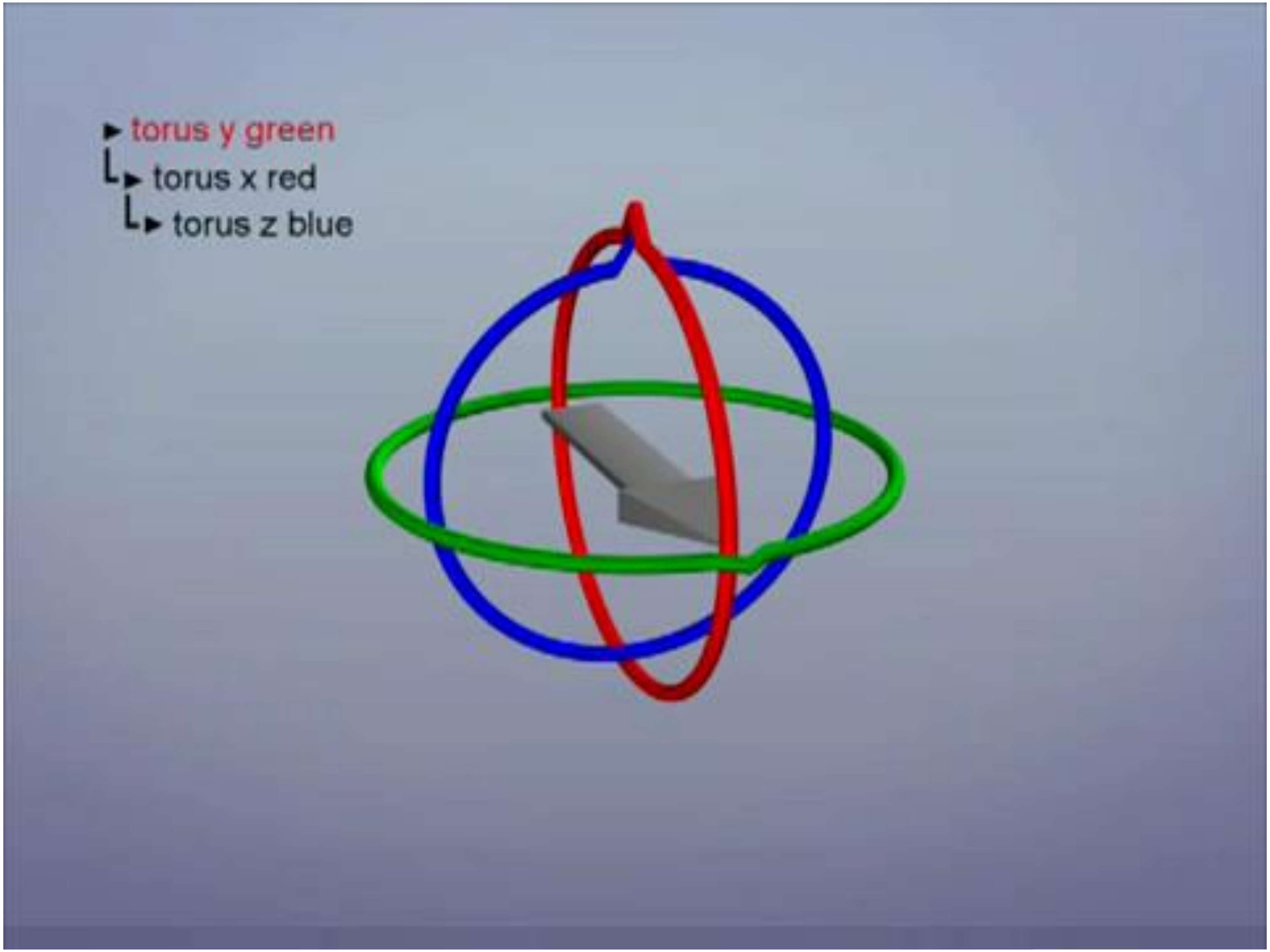
- There are no conventions, different orders in different industries, and order is often customizable (e.g., in software like Maya).
- There may be some practical differences between orderings and the best sequence may depend on what you are trying to accomplish.
  - In situations where there is a definite ground plane, Euler angles can actually be an intuitive representation (e.g., roll pitch yaw of a vehicle)



Based on slides by Steve Marschner

http://www.youtube.com/watch?v=zc8b2Jo7mno

Euler Angles – Gimbal Lock



Interpolating Euler Angles

- One can simply interpolate between the three values independently
- This will result in the interpolation following a different path depending on which of the 12 schemes you choose
- This may or may not be a problem, depending on your situation
- Interpolating near singularities is problematic
- Note: when interpolating angles, remember to check for crossing the +180/-180 degree boundaries

Euler Angles - Summary

- Euler angles are used in a lot of applications, but they tend to require some rather arbitrary decisions
- They also do not interpolate in a consistent way (but this isn’t always bad)
- They suffer from Gimbal lock and related problems
- There is no simple way to concatenate rotations
- Conversion to/from a matrix requires several trigonometry operations
- They are compact (requiring only 3 numbers)

Last Class?

Based on slides by Steve Marschner

Group of Unit Quaternions

- Quaternions are like complex numbers, but with three imaginary parts

$$ijk = i^2 = j^2 = k^2 = -1$$

- Example multiplication:

$$A = a_0 + a_1 i + a_2 j + a_3 k$$
$$B = b_0 + b_1 i + b_2 j + b_3 k$$
$$AB = a_0 b_0 - a_1 b_1 - a_2 b_2 - a_3 b_3 + (a_0 b_1 + a_1 b_0 + a_2 b_3 - a_3 b_2) i + \dots$$

The set of unit quaternions, combined with quaternion multiplication as a binary operation, form a group which is ***very similar*** to the rotation group SO(3), except that each 3D orientation appears twice! It is a double covering.

Quaternions as Rotations

- Related to a rotation by θ on unit length axis (x,y,z)

  $(c, sx, sy, sz) \equiv c + sxi + syj + szk$

  where $c = \cos(\theta/2)$ and $s = \sin(\theta/2)$
- Why $\theta/2$?
  - This means that θ and $-\theta$ are same rotation
  - The angle between two unit quaternions in 4D space is half the angle between the 3D orientations that they represent
- Composition by multiplication (rules on previous slide)
- Inverse of unit quaternions similar to complex conjugation
  - If $q = (w, x, y, z)$ is unit length, then $q^{-1} = (w, -x, -y, -z)$
  - Vector (x,y,z) transforms as $q\, v\, q^{-1}$ with $v = 0+ix+jy+kz$ and where the result is the imaginary part

Based on slides by Steve Marschner

Questions

- Cost of composing rotations:
  - Matrix? Quaternion?
- Cost of transforming vectors:
  - Matrix? Quaternion?

Based on slides by Steve Marschner

Questions

- Derive a matrix for a reflection in the plane with normal n going through the origin
- Derive a matrix for a reflection in the plane with normal n going through point $p_{0.}$

Comparison of Rotation representations

- Rotation matrix
  - 3x3 matrix with $R^T R = I$ and $\det(R) = +1$
- Euler angles
  - Used widely, often simple and convenient, also problematic
- Quaternion
  - Compact storage, efficient and nice mathematical properties…
- Axis and Angle
  - Similar to quaternions
- Axis scaled by angle
  - Save storage space!

Important issues for different rotation representations

- Storage
  - How much memory?
- Composition
  - Is it possible?
  - What's the cost?
  - What's the cost of transforming points and vectors?
- Conversion between representations?
  - Easy or Hard?

Numerical problems?
  - Suppose we need to compose hundreds or thousands of transformations?
  - Does this ever happen?
  - What can go wrong?
  - How can we fix it?
- Interpolation
  - Smoothly interpolating between two rotations is important for animation!

Linear Interpolation

- Linear interpolation between two points **a** and **b**

  $$\text{Lerp}(t,\mathbf{a},\mathbf{b}) = (1-t)\mathbf{a} + (t)\mathbf{b}$$

  where t ranges from 0 to 1
- Note that the Lerp operation can be thought of as a weighted average (convex)
- We can also write it in as an additive blend:

  $$\text{Lerp}(t,\mathbf{a},\mathbf{b}) = \mathbf{a} + t(\mathbf{b}-\mathbf{a})$$
- What happens if we use linear interpolation with rotations in different representations?

Based on slides by Steve Marschner

Spherical Linear Interpolation

- We define the spherical linear interpolation of two unit vectors in N dimensional space as

$$Slerp(t, \mathbf{a}, \mathbf{b}) = \frac{\sin\left(\left(1-t\right)\theta\right)}{\sin\theta}\mathbf{a} + \frac{\sin\left(t\theta\right)}{\sin\theta}\mathbf{b}$$

$$where: \theta = \cos^{-1}\left(\mathbf{a} \cdot \mathbf{b}\right)$$

- We can use this formula to smoothly interpolate two arbitrary rotations represented as **quaternions**.
  - Watch out if a dot b is negative!
  - What happens in this case? How to fix this?

Based on slides by Steve Marschner

Conversions for reference…

Axis Angle to Quaternion to Matrix

$$\mathbf{q} = \begin{bmatrix} \cos\frac{\theta}{2} & a_x \sin\frac{\theta}{2} & a_y \sin\frac{\theta}{2} & a_z \sin\frac{\theta}{2} \end{bmatrix}$$

or

$$\mathbf{q} = \left\langle \cos\frac{\theta}{2}, \mathbf{a}\sin\frac{\theta}{2} \right\rangle$$

$$\begin{bmatrix} 1-2q_2^2-2q_3^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & 1-2q_1^2-2q_3^2 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & 1-2q_1^2-2q_2^2 \end{bmatrix}$$

Based on slides by Steve Marschner

Axis Angle to Matrix

- Rotation θ around an arbitrary unit length axis a

$$\begin{bmatrix} a_x^2 + c_\theta(1-a_x^2) & a_x a_y(1-c_\theta) + a_z s_\theta & a_x a_z(1-c_\theta) - a_y s_\theta \\ a_x a_y(1-c_\theta) - a_z s_\theta & a_y^2 + c_\theta(1-a_y^2) & a_y a_z(1-c_\theta) + a_x s_\theta \\ a_x a_z(1-c_\theta) + a_y s_\theta & a_y a_z(1-c_\theta) - a_x s_\theta & a_z^2 + c_\theta(1-a_z^2) \end{bmatrix}$$

Euler Angles to Matrix

- To build a matrix from a set of Euler angles, we just multiply a sequence of rotation matrices together:

$$\mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_x & s_x \\ 0 & -s_x & c_x \end{bmatrix} \cdot \begin{bmatrix} c_y & 0 & -s_y \\ 0 & 1 & 0 \\ s_y & 0 & c_y \end{bmatrix} \cdot \begin{bmatrix} c_z & s_z & 0 \\ -s_z & c_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

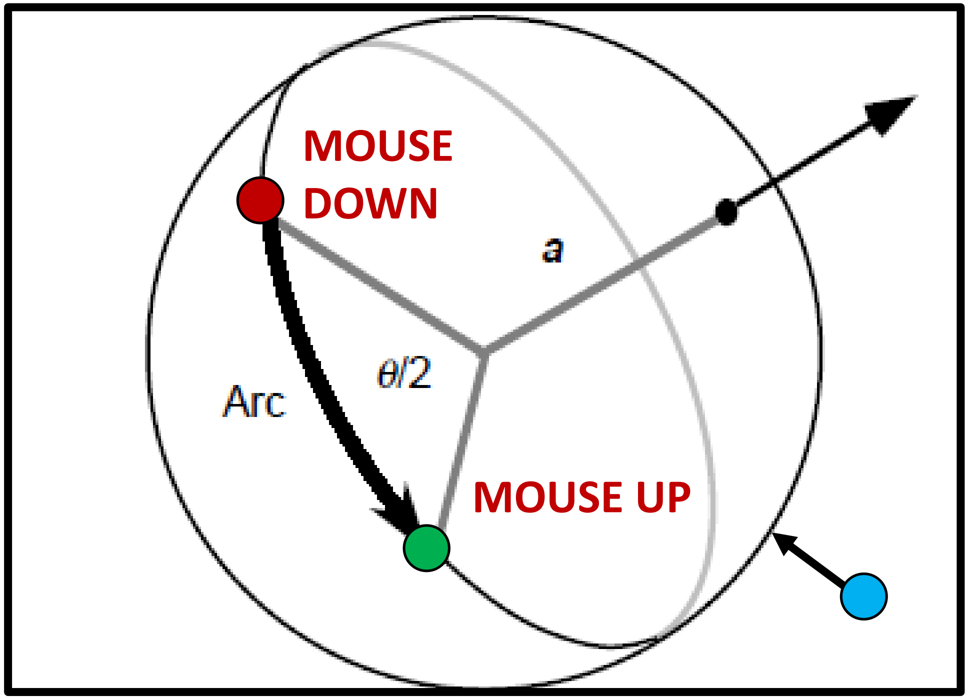
$$= \begin{bmatrix} c_y c_z & c_y s_z & -s_y \\ s_x s_y c_z - c_x s_z & s_x s_y s_z + c_x c_z & s_x c_y \\ c_x s_y c_z + s_x s_z & c_x s_y s_z - s_x c_z & c_x c_y \end{bmatrix}$$

Interaction

- What is a good way to rotate an object you are viewing on screen using a mouse?

ArcBall / TrackBall

Ken Shoemake, [ARCBALL: A User Interface for Specifying Three-Dimensional Orientation Using a Mouse](), Graphics Interface 1992.



- **Fit** specifies size of the ball, radius = min screen dim / fit, 2 means just touching edges.
- **Gain** specifies a modification to the computed arc angle.

- Imagine: ball centered on the screen
- screen is at z=0 with axis pointing out of the screen
- Dragging the mouse from one screen point to another rotates the point on the ball.
- **How do we compute the rotation?**
  - Axis?
  - Angle?
- When mouse not on ball, project onto ball.

Coming up with a rotation matrix

- We have seen matrices for coordinate axis rotations, and we have seen formulas for quaternion and axis angle to matrix on previous slides…
  - What if we want rotation about some random axis? But with an intuitive / constructive solution…
- Compute by composing elementary transforms
  - transform rotation axis to align with *x* axis
  - apply rotation
  - inverse transform back into position
- Just as in 2D this can be interpreted as a similarity transform

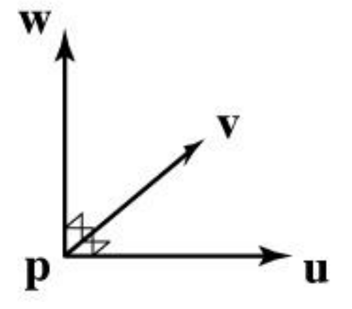
Building general rotations

- Using elementary transforms you need three
  - translate axis to pass through origin
  - rotate about *y* to get into *x*-*y* plane
  - rotate about *z* to align with *x* axis
- Alternative: construct frame and change coordinates
  - choose *p*, *u*, *v*, *w* to be orthonormal frame with *p* and *u* matching the rotation axis
  - apply similarity transform $T = F\, R_x(\theta)\, F^{-1}$

Orthonormal frames in 3D

- Useful tools for constructing transformations
- Recall rigid motions
  - affine transforms with pure rotation
  - columns (and rows) form right handed **o**rtho**n**ormal **b**asis

$$F = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Based on slides by Steve Marschner

Building 3D frames

- Given a vector **a** and a secondary vector **b**
  - The **u** axis should be parallel to **a**; the **u**–**v** plane should contain **b**
    - **u** = **u** / ||**u**||
    - **w** = **u** x **b**; **w** = **w** / ||w||
    - **v** = **w** x **u**
- Given just a vector **a**
  - The **u** axis should be parallel to **a**; don't care about orientation about that axis
    - Same process but choose arbitrary **b** first
    - Good choice is not near **a**, e.g., set smallest entry to 1

Building general rotations

- Alternative: construct frame and change coordinates
  - choose *p*, *u*, *v*, *w* to be orthonormal frame with *p* and *u* matching the rotation axis
  - apply similarity transform $T = F\, R_x(\theta)\, F^{-1}$
  - interpretation: move to *x* axis, rotate, move back
  - interpretation: rewrite *u*-axis rotation in new coordinates
  - (each is equally valid)

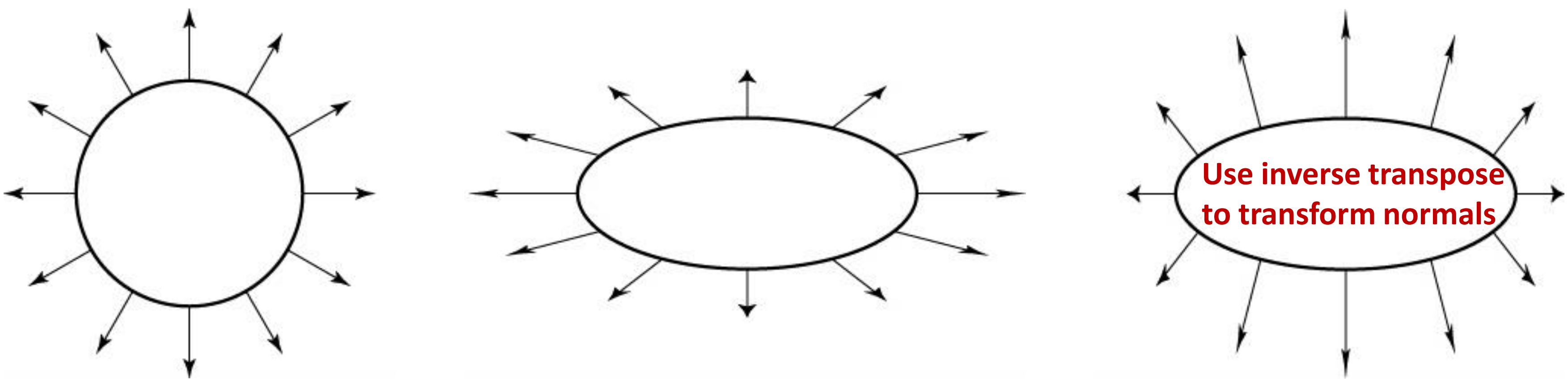
Based on slides by Steve Marschner

Building transforms from points

- 2D affine transformations have 6 degrees of freedom (DOFs)
  - this is the number of “knobs” we have to set to define one
- Therefore 6 constraints suffice to define the transformation
  - Constrain point p maps to point q (2 constraints at once)
  - three point constraints add up to constrain all 6 DOFs
    (i.e., can map any triangle to any other triangle)
- 3D affine transformation has 12 degrees of freedom
  - count them by looking at the matrix entries we’re allowed to change
- Therefore 12 constraints suffice to define the transformation
  - in 3D, this is 4 point constraints
    (i.e., can map any tetrahedron to any other tetrahedron)

Transforming normal vectors

- Transforming surface normals
  - differences of points (e.g., tangents) transform OK
  - normals do not



$$
\begin{aligned}
&\text{have: } \mathbf{t} \cdot \mathbf{n} = \mathbf{t}^T \mathbf{n} = 0 \\
&\text{want: } M\mathbf{t} \cdot X\mathbf{n} = \mathbf{t}^T M^T X \mathbf{n} = 0 \\
&\text{so set } X = (M^T)^{-1} \\
&\text{then: } M\mathbf{t} \cdot X\mathbf{n} = \mathbf{t}^T M^T (M^T)^{-1} \mathbf{n} = \mathbf{t}^T \mathbf{n} = 0
\end{aligned}
$$

Question

Let $p = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)^T$ be a point in non-homogeneous coordinates on a sphere with center at the origin and with radius equal to 1. Let $n = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)^T$ be the normal vector in non-homogeneous coordinates of the sphere at point p. Given transformations, T, S, R,

$$T = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

(a) What is the position of the point p on the sphere after the sphere is transformed by the product TSR? Show your work by appling each transformation separately rather than computing the product of the three matrices.

(b) What is the normal of the point p on the sphere after the sphere is transformed by the product TSR? Show your work by appling each transformation separately.

Review and more information

- Textbook (3rd edition)
  - 6.1 - 2D linear transformations, composition
  - 6.2 - 3D linear transformations, transforming normal vectors
  - 6.3 - Translation and affine transformations, homogeneous coordinates
  - 6.5 - Coordinate transformations
- Rotations not covered in much depth in the textbook, but note in particular:
  - 6.2.1 general rotations
  - 17.2.2 Interpolating rotations (gimbal lock, quaternions)