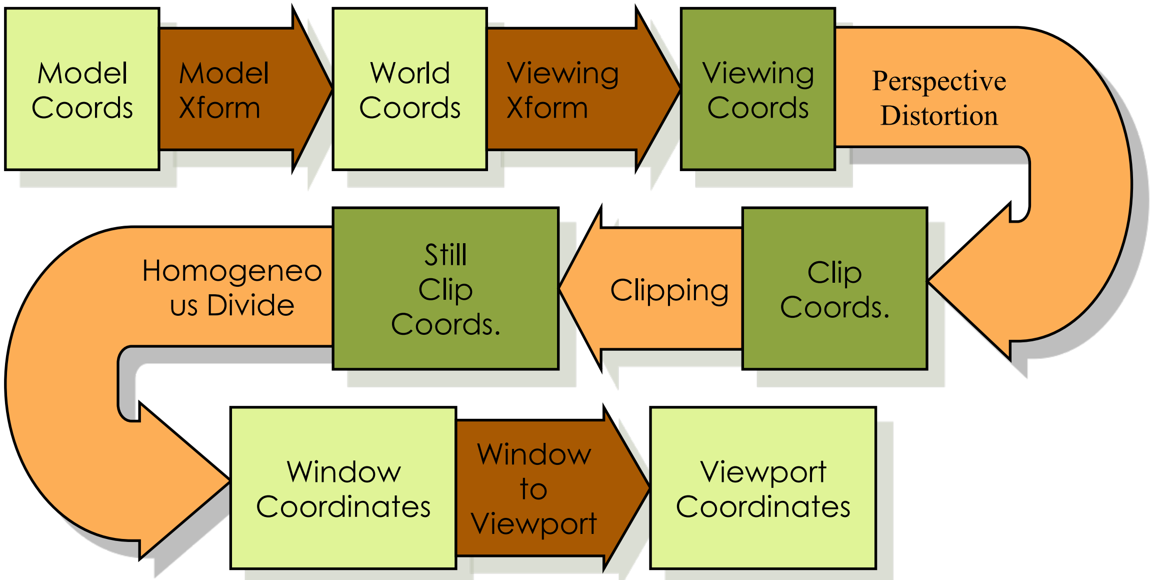
CS 418: Interactive Computer Graphics

Introduction to Quaternions

Eric Shaffer

Everything we have learned in one slide




```
attribute vec3 aVertexPosition;
uniform mat4 uMVMatrix;
Uniform mat4 uPMatrix;
 void main(void) {
       gl_Position =uPMatrix*uMVMatrix*vec4(aVertexPosition, 1.0);
}
```

Representing Orientations is Complex

- No simple means of representing a 3D orientation
  - unlike position and Cartesian coordinates
- There are several popular options:
  - Euler angles
  - Rotation vectors (axis/angle)
  - 3x3 matrices
  - Quaternions

Euler Angles

- We can represent an orientation with 3 numbers
- A sequence of rotations around principal axis is called an *Euler Angle Sequence*
- Assuming we limit ourselves to 3 rotations without successive rotations about the same axis, we could use any of the following 12 sequences:

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

Euler Angles to Matrix Conversion

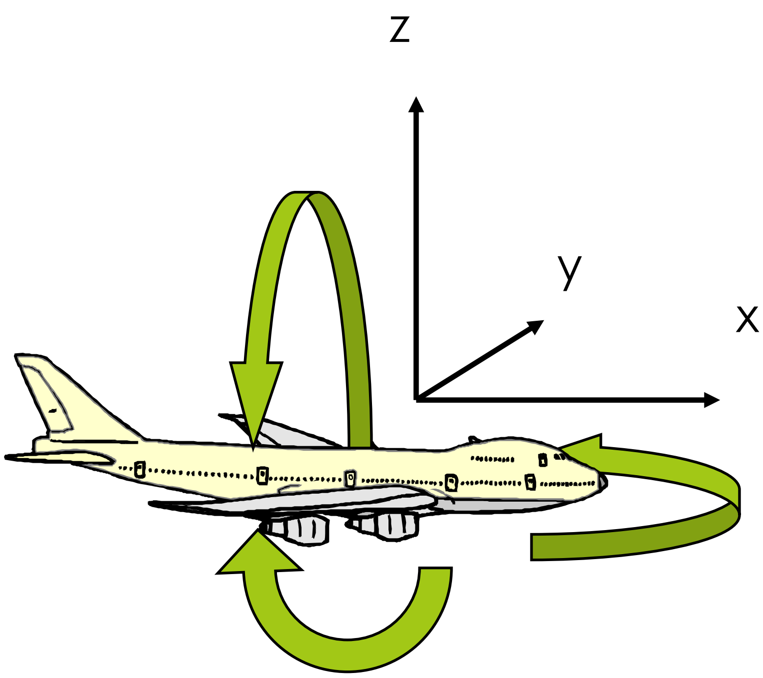
- To build a matrix from a set of Euler angles, we just multiply a sequence of rotation matrices together:

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

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

Euler Angles

- Airplane orientation
  - Roll
    - rotation about x
    - Turn wheel
  - Pitch
    - rotation about y
    - Push/pull wheel
  - Yaw
    - rotation about z
    - Rudder (foot pedals)
- Airplane orientation
  - Rx(roll) Ry(pitch) Rz(yaw)



Rotation Order with Euler Angles

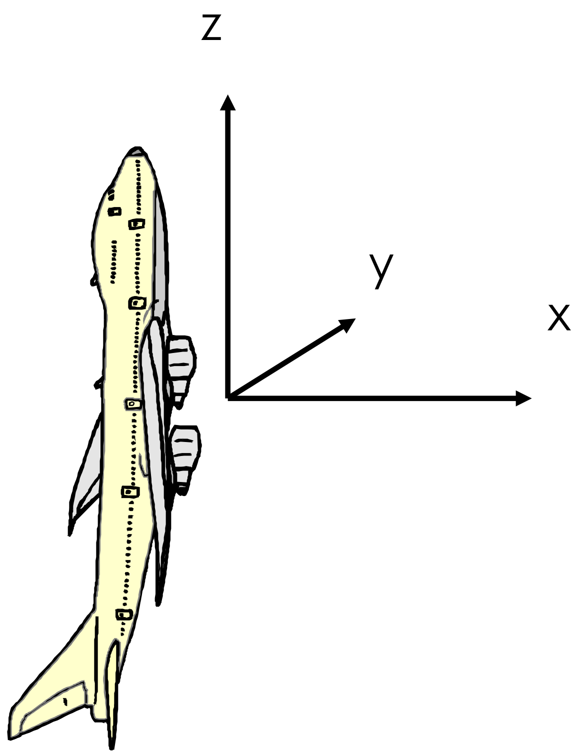
- Usually: $R_x R_y R_z$
- So, around Z first
- Then around Y_1
  - Around $R_z <0,1,0>$
- Then around X_2
  - $R_y R_z <1,0,0>$
- Think of rotations occurring in three different frames
  - XYZ then $X_1 Y_1 Z_1$ and then $X_2 Y_2 Z_2$

Euler Angles

- Euler angles are used in a lot of applications
  - Intuitive…
- They are compact (requiring only 3 numbers)
- Ambiguous: different triples can be same orientation
- Ambiguous: Order of matrix multiplication will affect the result
- They do not interpolate in a obvious way
- They can suffer from Gimbal lock
- There is no simple way to concatenate rotations
- Conversion to/from a matrix requires several trig operations

Gimbal Lock

- Airplane orientation
  - Rx(roll) Ry(pitch) Rz(yaw)
- Two axes have collapsed onto each other

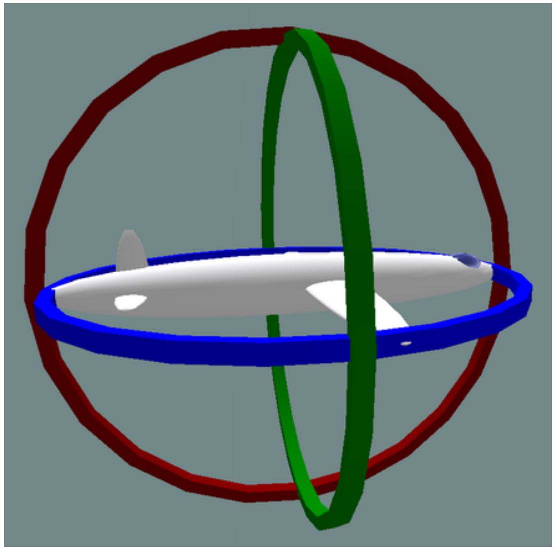


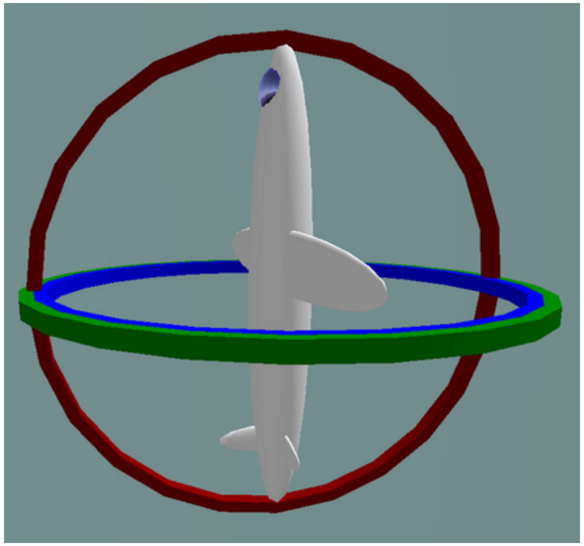
Gimbal Lock

Problem if your camera view is a sequence of rotation matrices

User input to rotate around Y and Z do the same thing in the example below

Can break a user interface….

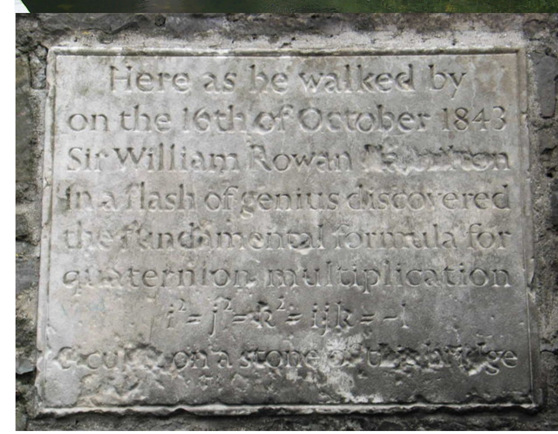




Quaternions

- Developed by Sir William Rowan Hamilton [1843]
- Quaternions are 4-D numbers
- With one real axis
- And three imaginary axes: **i**,**j**,**k**
- Imaginary multiplication rules

$$\mathbf{q} = q_0 + iq_1 + jq_2 + kq_3$$



Hamilton Math Inst., Trinity College

Quaternions

- Introduced to Computer Graphics by Shoemake [1985]
- Given an angle and axis, easy to convert to and from quaternion
  - Euler angle conversion to and from arbitrary axis and angle difficult
- Quaternions allow stable and constant interpolation of orientations
  - Cannot be done easily with Euler angles

Unit Quaternions

- For convenience, we will use only unit length quaternions, as they will be sufficient for our purposes and make things a little easier

$$|\mathbf{q}| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1$$

- These correspond to the set of vectors that form the 'surface' of a 4D hypersphere of radius 1

Quaternions as Rotations

- A quaternion can represent a rotation by an angle θ around a unit vector **a**:

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

or

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

- If **a** is unit length, then **q** will be also

Quaternions as Rotations

$$
\begin{aligned}
|\mathbf{q}| &= \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \\
&= \sqrt{\cos^2\frac{\theta}{2} + a_x^2 \sin^2\frac{\theta}{2} + a_y^2 \sin^2\frac{\theta}{2} + a_z^2 \sin^2\frac{\theta}{2}} \\
&= \sqrt{\cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}\left(a_x^2 + a_y^2 + a_z^2\right)} \\
&= \sqrt{\cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}|\mathbf{a}|^2} = \sqrt{\cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}} \\
&= \sqrt{1} = 1
\end{aligned}
$$

Rotation using Quaternions

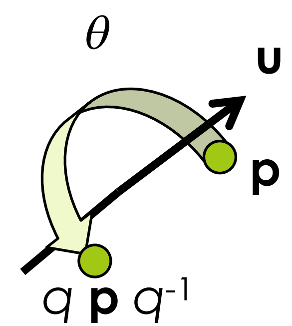
- Let $q = \cos(\theta/2) + \sin(\theta/2)\ \mathbf{u}$ be a unit quaternion: $|q| = |\mathbf{u}| = 1$.
- Let point $\mathbf{p} = (x,y,z) = x\ \mathbf{i} + y\ \mathbf{j} + z\ \mathbf{k}$
- Then the product $q\ \mathbf{p}\ q^{-1}$ rotates the point $\mathbf{p}$ about axis $\mathbf{u}$ by angle θ
- Inverse of a unit quaternion is its conjugate (negate the imaginary part)

  $$\begin{aligned} q^{-1} &= (\cos(\theta/2) + \sin(\theta/2)\ \mathbf{u})^{-1} \\ &= \cos(-\theta/2) + \sin(-\theta/2)\ \mathbf{u} \\ &= \cos(\theta/2) - \sin(\theta/2)\ \mathbf{u} \end{aligned}$$

- Composition of rotations

$$q_{12} = q_1\, q_2 \neq q_2\, q_1$$

$$q = \cos\frac{\theta}{2} + \mathbf{u}\sin\frac{\theta}{2}$$



Quaternion to Matrix

- To convert a quaternion to a rotation matrix:

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

Matrix to Quaternion

- Matrix to quaternion is not hard
  - It involves a few ‘if’ statements,
  - a square root,
  - three divisions,
  - and some other stuff
- tr(M) is the trace
  - sum of the diagonal elements

$$q_0 = \frac{1}{2}\sqrt{1 + tr(\mathbf{M})}$$

$$q_1 = \frac{m_{21} - m_{12}}{4q_0}$$

$$q_2 = \frac{m_{02} - m_{20}}{4q_0}$$

$$q_3 = \frac{m_{10} - m_{01}}{4q_0}$$

Quaternion Dot Products

- The dot product of two quaternions works in the same way as the dot product of two vectors:

$$\mathbf{p} \cdot \mathbf{q} = p_0 q_0 + p_1 q_1 + p_2 q_2 + p_3 q_3 = |\mathbf{p}||\mathbf{q}| \cos\varphi$$

- The angle between two quaternions in 4D space is half the angle one would need to rotate from one orientation to the other in 3D space

Quaternion Multiplication

- We can perform multiplication on quaternions if we expand them into their complex number form $\mathbf{q} = q_0 + iq_1 + jq_2 + kq_3$
- If **q** represents a rotation and **q'** represents a rotation, then **qq'** represents **q** rotated by **q'**
- This follows very similar rules as matrix multiplication (I.e., non-commutative)

$$\mathbf{qq}' = \left(q_0 + iq_1 + jq_2 + kq_3\right)\left(q_0' + iq_1' + jq_2' + kq_3'\right)$$
$$= \left\langle ss' - \mathbf{v}\cdot\mathbf{v}', s\mathbf{v}' + s'\mathbf{v} + \mathbf{v}\times\mathbf{v}'\right\rangle$$

Quaternion Multiplication

- Note that two unit quaternions multiplied together will result in another unit quaternion
- This corresponds to the same property of complex numbers
- Remember that multiplication by complex numbers can be thought of as a rotation in the complex plane
- Quaternions extend the planar rotations of complex numbers to 3D rotations in space

Linear Interpolation

- If we want to do a linear interpolation between two points **a** and **b** in normal space

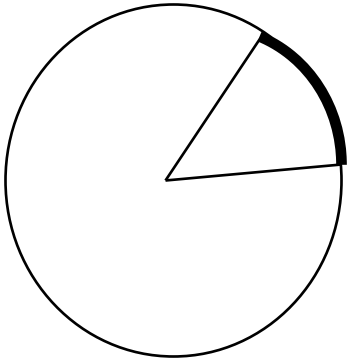
$$\mathrm{Lerp}(t,\mathbf{a},\mathbf{b}) = (1-t)\mathbf{a} + (t)\mathbf{b}$$

  where t ranges from 0 to 1
- Note that the Lerp operation can be thought of as a weighted average (convex)
- We could also write it in it's additive blend form:

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

Spherical Linear Interpolation

- If we want to interpolate between two points on a sphere (or hypersphere), we don't just want to Lerp between them
- Instead, we will travel across the surface of the sphere by following a 'great arc'



Spherical Linear Interpolation

- We define the spherical linear interpolation of two unit quaternions **a** and **b** as:

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

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

Quaternion Interpolation

- Useful for animating objects between two poses
- Not useful for all camera orientations
  - up vector can become tilted and annoy viewers
  - depends on application
- Interpolated path through SLERP rotates
  - around a fixed axis
  - at a constant speed
  - so, no acceleration

If we want to interpolate through a series of orientations **q1,q2,…,qn** is SLERP a good choice?