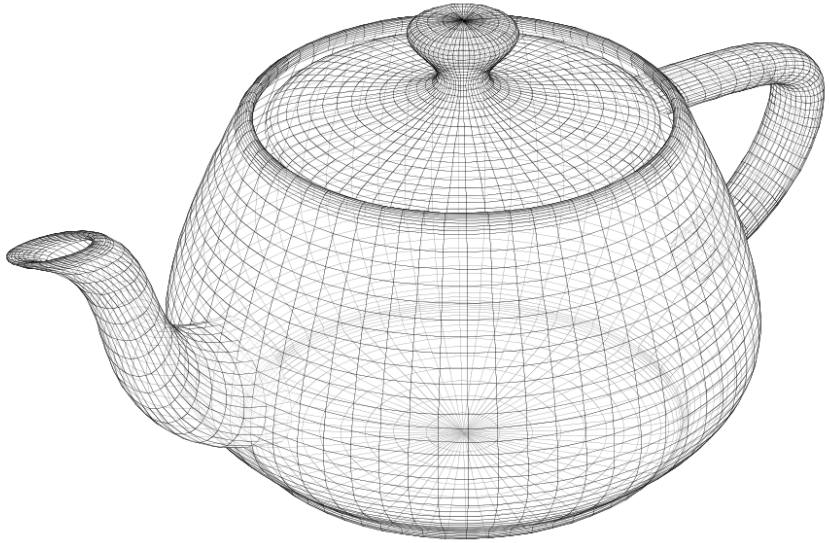
# Mathematics for Orientation

# Quaternions



CS 418: Interactive Computer Graphics

Professor Eric Shaffer

*If you do not change direction,*
*you may end up where you are heading.*

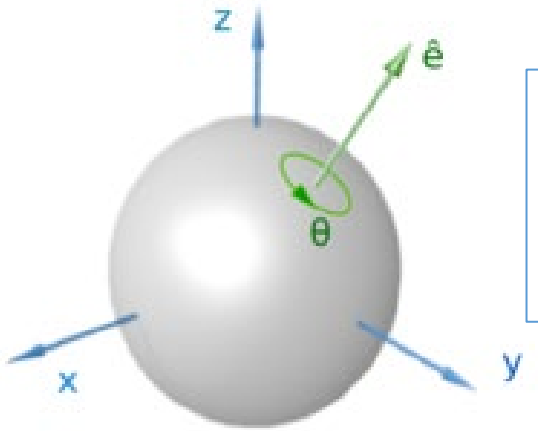
— Lao Tzu (600–531 BCE)

ILLINOIS

# Euler’s Rotation Theorem (1775)

In geometry, **Euler's rotation theorem** states that, in three-dimensional space, any displacement of a rigid body such that a point on the rigid body remains fixed, is equivalent to a single rotation about some axis that runs through the fixed point. It also means that the composition of two rotations is also a rotation.

--Wikipedia

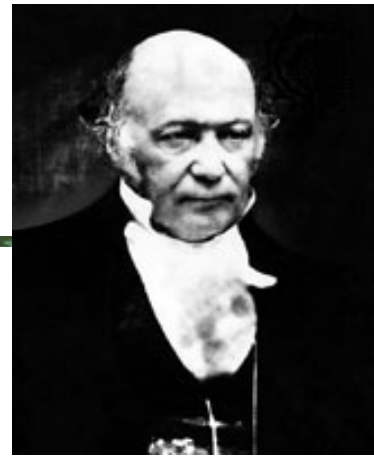



Is an angle-axis representation of an orientation unique?

This means that any set of Euler angles is equivalent to some single rotation around some axis. So, any orientation can be represented by and angle and an axis (i.e. a vector)

ILLINOIS

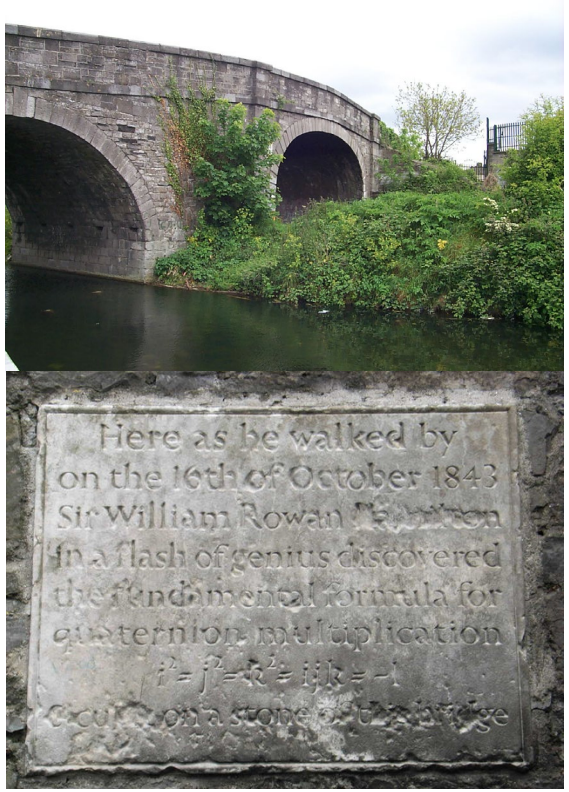
# Quaternions



- Alternative to Euler Angles
- Developed by Sir William Rowan Hamilton [1843]
- Quaternions are 4-D complex numbers
- With one real axis
- And three imaginary axes: **i,j,k**

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

TL;DL It turns out that quaternions are effectively an angle-axis representation of an orientation…just think of them as a way of encoding that information




Hamilton Math Inst., Trinity College

ILLINOIS

# 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

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

- These correspond to the set of 4D vectors
- They form the ‘surface’ of a 4D hypersphere of radius 1

# Quaternions as Rotations

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

$$\mathbf{q} = \left[ \cos\frac{\theta}{2} \quad a_x \sin\frac{\theta}{2} \quad a_y \sin\frac{\theta}{2} \quad a_z \sin\frac{\theta}{2} \right]$$

*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

# A Rotation Quaternion is Unit-Length

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

$$= \sqrt{\cos^2\frac{\theta}{2} + a_x^2 \sin^2\frac{\theta}{2} + a_y^2 \sin^2\frac{\theta}{2} + a_z^2 \sin^2\frac{\theta}{2}}$$

$$= \sqrt{\cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}\left(a_x^2 + a_y^2 + a_z^2\right)}$$

$$= \sqrt{\cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}|\mathbf{a}|^2} = \sqrt{\cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}}$$

$$= \sqrt{1} = 1$$

ILLINOIS

# 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 $\theta$
- Inverse of a unit quaternion is its ***conjugate***

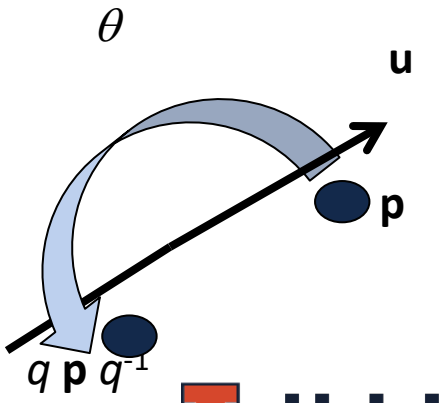
  **…just negate the imaginary part**

  $q^{-1} = (\cos(\theta/2) + \sin(\theta/2)\,\mathbf{u})^{-1}$
  $= \cos(-\theta/2) + \sin(-\theta/2)\,\mathbf{u}$
  $= \cos(\theta/2) - \sin(\theta/2)\,\mathbf{u}$

- Composition of rotations $q_{12} = q_1\,q_2 \neq q_2\,q_1$

We haven't talked about how to multiply quaternions yet, but don't worry about that for now…

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




ILLINOIS

# Quaternion to Matrix

- To convert a quaternion to a rotation matrix:

Again, why do we want to be able to do this?

Notice – no trig functions involved so it is fast

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

ILLINOIS

# 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

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

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

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

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

This assumes M is a 4x4 homogeneous rotation matrix…so the diagonal ends with a 1

# Matrix to Quaternion…Numerically Stable

Division by small magnitudes should be avoided….

Determine if $q_0$ is largest component using

$$m_{00} = t + 2q_1^2$$
$$m_{11} = t + 2q_2^2$$
$$m_{22} = t + 2q_3^2$$
$$m_{00} + m_{11} + m_{22} = t + 2q_0^2$$

Pick largest $q_i$ based on the above and compute it using:

$$4q_1^2 = m_{00} - m_{11} - m_{22} + m_{33}$$
$$4q_2^2 = -m_{00} + m_{11} - m_{22} + m_{33}$$
$$4q_3^2 = -m_{00} - m_{11} + m_{22} + m_{33}$$
$$4q_0^2 = m_{00} + m_{11} + m_{22} + m_{33}$$

Then compute the other values using:

$$m_{21} - m_{12} = 4q_0q_1$$
$$m_{02} - m_{20} = 4q_0q_2$$
$$m_{10} - m_{01} = 4q_0q_3$$

**Numerical stability** refers to how a malformed input affects the execution of an algorithm. In a **numerically stable** algorithm, errors in the input lessen in significance as the algorithm executes, having little effect on the final output.
-wikipedia

ILLINOIS

# Quaternion Dot Products

The dot product of two quaternions:

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

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

ILLINOIS

# Quaternion Multiplication

- We can perform multiplication on quaternions
  - 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, **qq′** represents **q** rotated by **q′**
- This follows very similar rules as matrix multiplication (i.e., non-commutative)

$$\mathbf{qq}' = \left(q_0 + iq_1 + jq_2 + kq_3\right)\left(q_0' + iq_1' + jq_2' + kq_3'\right)$$

$$= \left\langle ss' - \mathbf{v}\cdot\mathbf{v}', s\mathbf{v}' + s'\mathbf{v} + \mathbf{v}\times\mathbf{v}'\right\rangle$$

It's just like multiplying 2 rotation matrices together….

ILLINOIS

# Quaternion Multiplication

- Two unit quaternions multiplied together results in another unit quaternion
- This corresponds to the same property of complex numbers
- Remember(?) multiplication by complex numbers is like 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

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

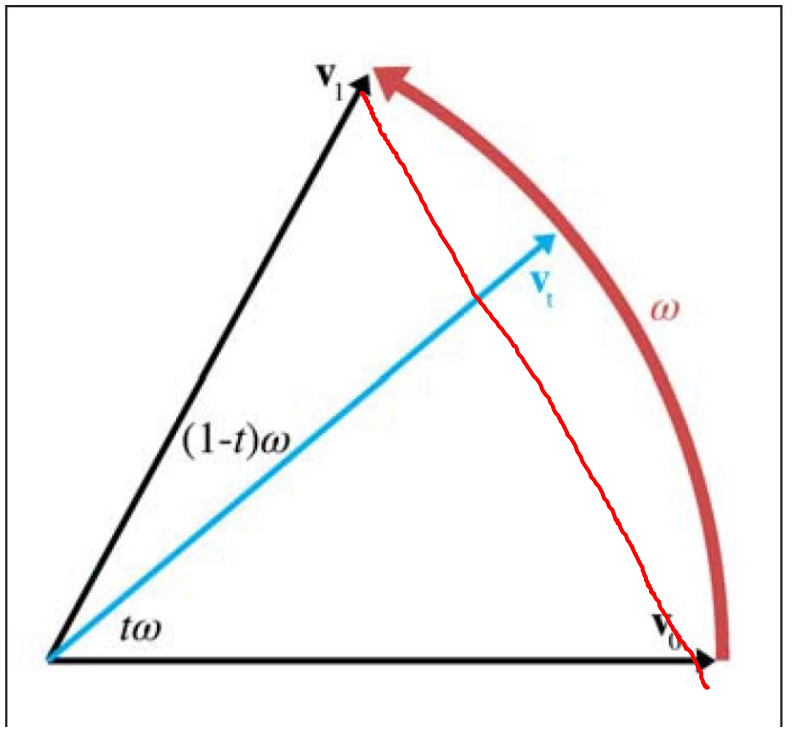
  where t ranges from 0 to 1
- Note that the Lerp operation can be thought of as a weighted average (convex)
- We could also write it in its additive blend form:

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

# Spherical Linear Interpolation

- If we want to interpolate between two points on a sphere (or hypersphere), we do not just want to Lerp between them

- Instead, we will travel across the surface of the sphere by following a ‘great arc’




If we lerped between 2 unit quaternions would our interpolated quaternions be valid orientations?

ILLINOIS

# Spherical Linear Interpolation

- The spherical linear interpolation of two unit quaternions **a** and **b** is:

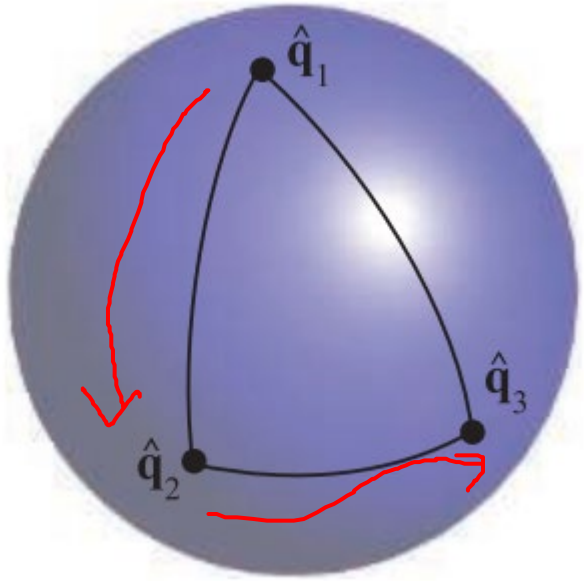
$$Slerp(t,\mathbf{a},\mathbf{b}) = \frac{\sin((1-t)\theta)}{\sin\theta}\mathbf{a} + \frac{\sin(t\theta)}{\sin\theta}\mathbf{b}$$

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

Quick quiz: explain what the angle theta is? What space is it in?

# Quaternion Interpolation

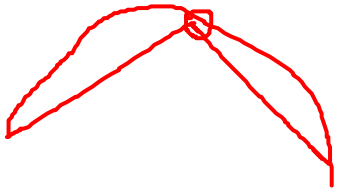
- Useful for animating objects between two poses
- Not useful for all camera orientations
  - up vector can become tilted and annoy viewers
  - depends on application
- Interpolated path through SLERP rotates
  - around a fixed axis
  - at a constant speed
  - so, no acceleration




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

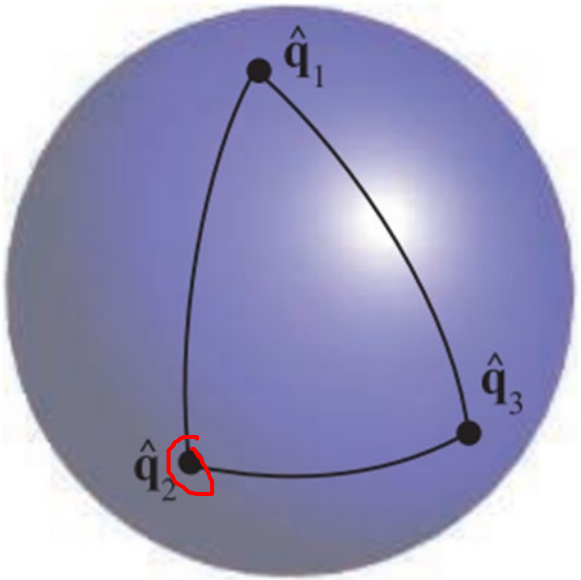
ILLINOIS

# The squad function



$slerp$ing $\hat{q}_1 \rightarrow \hat{q}_2 \rightarrow \hat{q}_3$ here will be a sudden jerk in the orientation at $\hat{q}_2$

Can perform spherical cubic interpolation instead.
Between each pair of quaternions introduce two control quaternions $\hat{a}_i, \hat{a}_{i+1}$




$$\hat{\mathbf{a}}_i = \hat{\mathbf{q}}_i \exp\left[-\frac{\log(\hat{\mathbf{q}}_i^{-1}\hat{\mathbf{q}}_{i-1}) + \log(\hat{\mathbf{q}}_i^{-1}\hat{\mathbf{q}}_{i+1})}{4}\right]$$

$$\mathtt{squad}(\hat{\mathbf{q}}_i, \hat{\mathbf{q}}_{i+1}, \hat{\mathbf{a}}_i, \hat{\mathbf{a}}_{i+1}, t) =$$
$$\mathtt{slerp}(\mathtt{slerp}(\hat{\mathbf{q}}_i, \hat{\mathbf{q}}_{i+1}, t), \mathtt{slerp}(\hat{\mathbf{a}}_i, \hat{\mathbf{a}}_{i+1}, t), 2t(1-t))$$

Interpolation will not pass through the control quaternions.
It will smoothly move from one orientation to another.

ILLINOIS