Animation

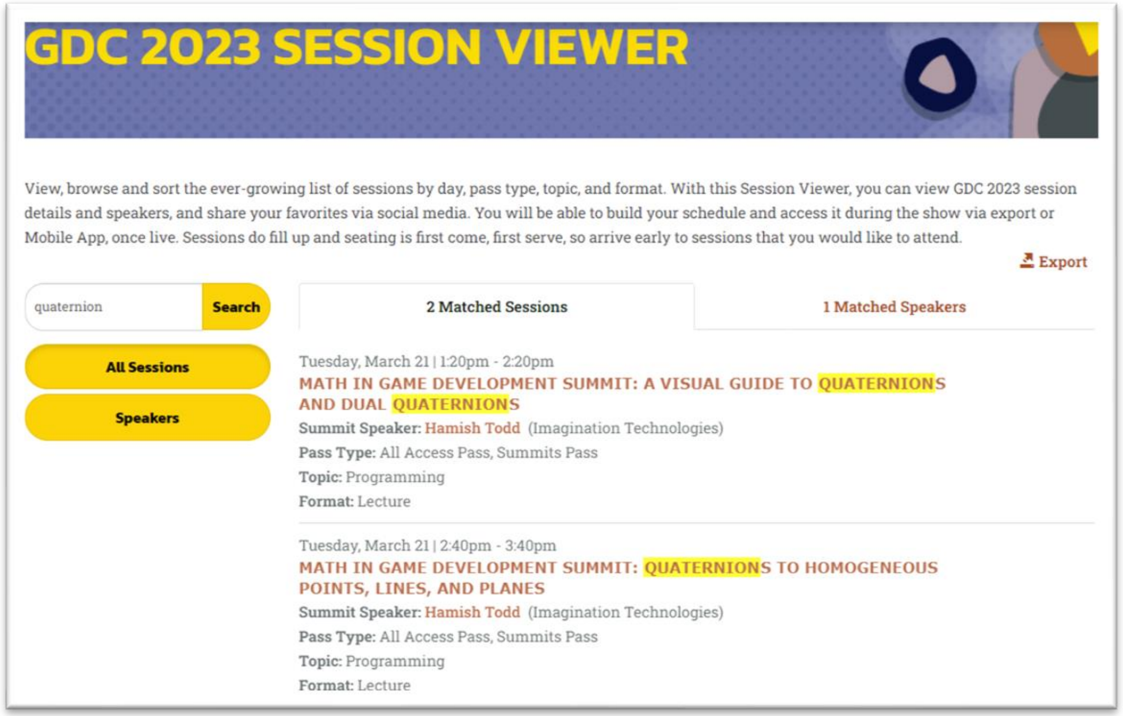
Orientation: Quaternions

CS 415: Game Development

Professor Eric Shaffer

Quaternions

- Alternative way of encoding orientation
- Game industry standard
- Both Unity and Unreal use them
- User interface may still use Euler Angles
  - Converted to quaternions under the hood



GDC 2023 SESSION VIEWER

View, browse and sort the ever-growing list of sessions by day, pass type, topic, and format. With this Session Viewer, you can view GDC 2023 session details and speakers, and share your favorites via social media. You will be able to build your schedule and access it during the show via export or Mobile App, once live. Sessions do fill up and seating is first come, first serve, so arrive early to sessions that you would like to attend.

Tuesday, March 21 | 1:20pm - 2:20pm
MATH IN GAME DEVELOPMENT SUMMIT: A VISUAL GUIDE TO QUATERNIONS AND DUAL QUATERNIONS
Summit Speaker: Hamish Todd (Imagination Technologies)
Pass Type: All Access Pass, Summits Pass
Topic: Programming
Format: Lecture

Tuesday, March 21 | 2:40pm - 3:40pm
MATH IN GAME DEVELOPMENT SUMMIT: QUATERNIONS TO HOMOGENEOUS POINTS, LINES, AND PLANES
Summit Speaker: Hamish Todd (Imagination Technologies)
Pass Type: All Access Pass, Summits Pass
Topic: Programming
Format: Lecture

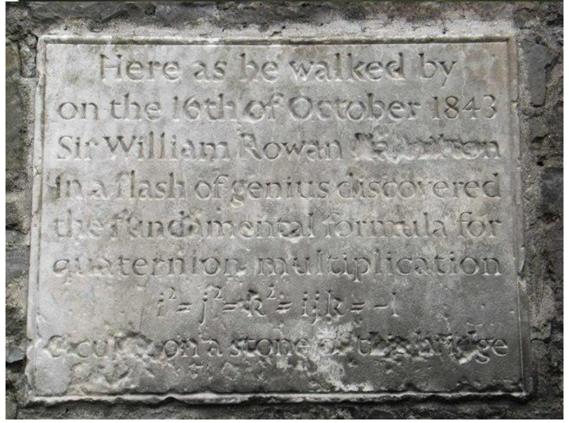
ILLINOIS

Quaternions

- 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


Here as he walked by
on the 16th of October 1843
Sir William Rowan Hamilton
in a flash of genius discovered
the fundamental formula for
quaternion multiplication
$i^2 = j^2 = k^2 = ijk = -1$
& cut it on a stone of this bridge

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

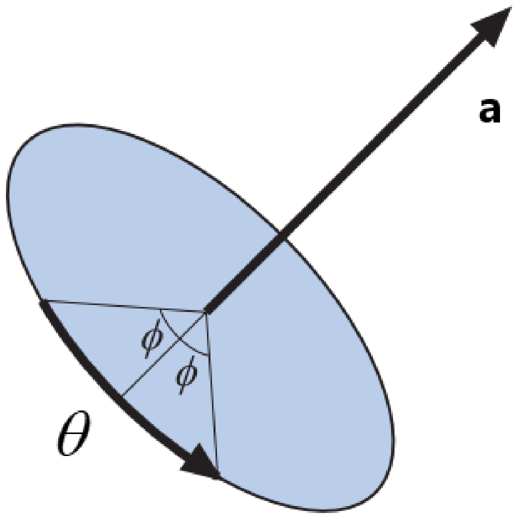
Quaternions as Rotations

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

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

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

Rotation using Quaternions

We will almost never do this…next slide explains why….

- 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

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

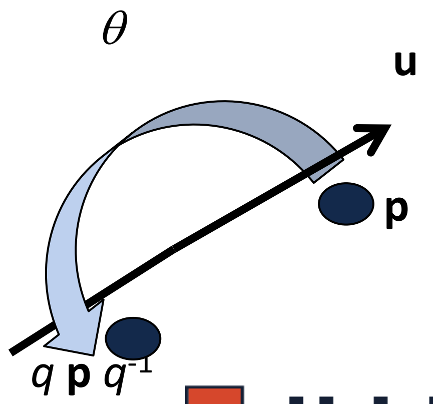
- Inverse of a unit quaternion is its ***conjugate***

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

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



ILLINOIS

Quaternion to Matrix

- To convert a quaternion to a rotation matrix:

Why do we want to be able to do this?

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

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

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

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

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

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

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

Quaternion Dot Products

The dot product of two quaternions:

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

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
  - we expand them into their complex number form

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

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

- If **q**=(s,**v**) represents a rotation and **q'**=(s',**v'**) represents a rotation,

  qq' represents **q** rotated by **q'**

- This follows very similar rules as matrix multiplication (i.e., non-commutative)

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

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

Quaternion Multiplication

- 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

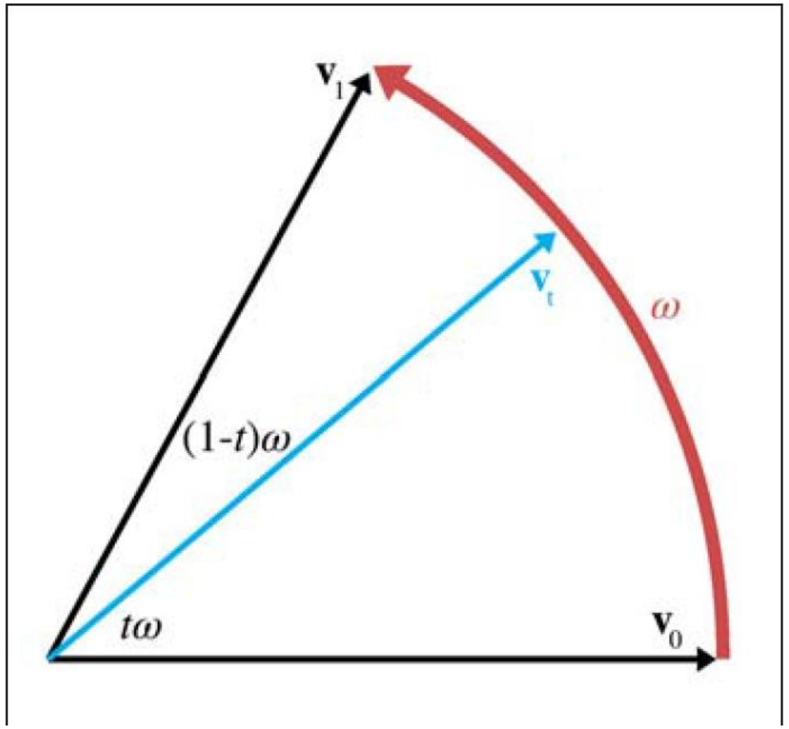
- Note that the Lerp operation can be thought of as a weighted average (convex)
- We could also write it in its additive blend form:

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

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?

Spherical Linear Interpolation

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

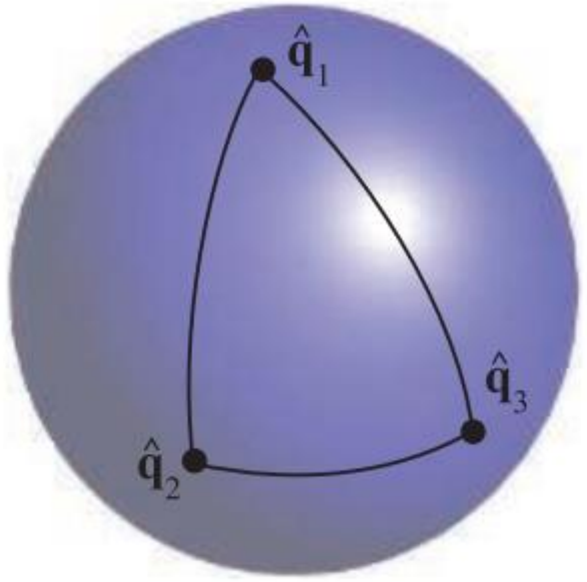
$$Slerp(t,\mathbf{a},\mathbf{b}) = \frac{\sin((1-t)\theta)}{\sin\theta}\mathbf{a} + \frac{\sin(t\theta)}{\sin\theta}\mathbf{b}$$

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

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

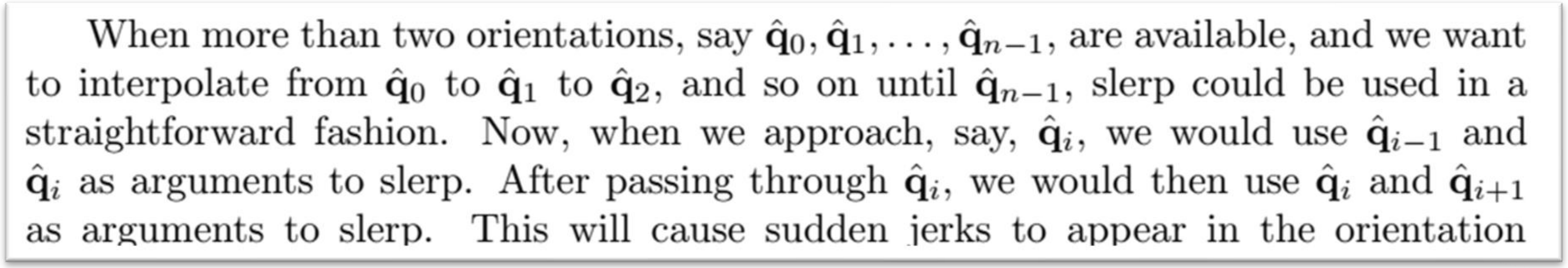
Quaternion Interpolation

- 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



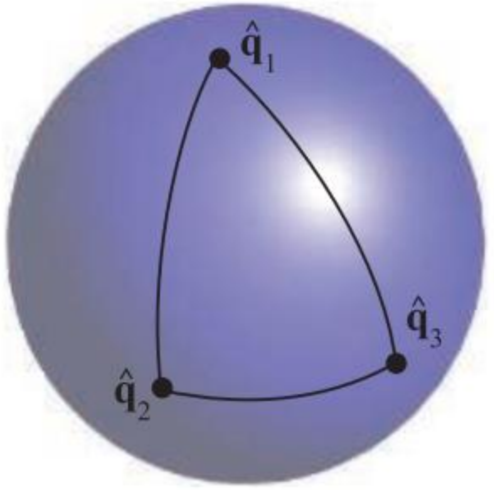
If we want to interpolate through a series of orientations **q1,q2,…,qn** is SLERP a good choice?

Chained Quaternion Interpolation

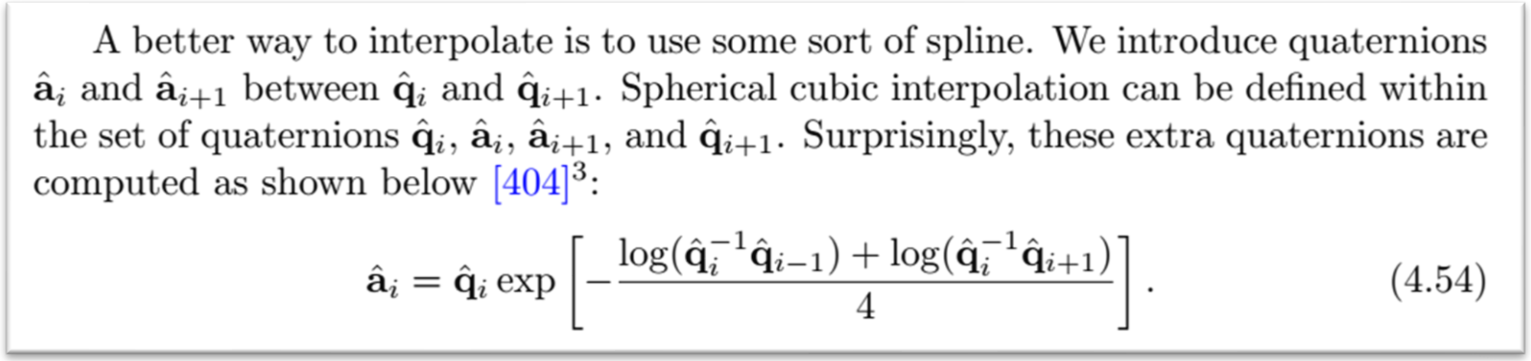


When more than two orientations, say $\hat{\mathbf{q}}_0, \hat{\mathbf{q}}_1, \ldots, \hat{\mathbf{q}}_{n-1}$, are available, and we want to interpolate from $\hat{\mathbf{q}}_0$ to $\hat{\mathbf{q}}_1$ to $\hat{\mathbf{q}}_2$, and so on until $\hat{\mathbf{q}}_{n-1}$, slerp could be used in a straightforward fashion. Now, when we approach, say, $\hat{\mathbf{q}}_i$, we would use $\hat{\mathbf{q}}_{i-1}$ and $\hat{\mathbf{q}}_i$ as arguments to slerp. After passing through $\hat{\mathbf{q}}_i$, we would then use $\hat{\mathbf{q}}_i$ and $\hat{\mathbf{q}}_{i+1}$ as arguments to slerp. This will cause sudden jerks to appear in the orientation

Akenine-Moeller, Tomas; Haines, Eric; Hoffman, Naty. Real-Time Rendering



Can smooth the path by using a spherical curve instead of straight line



A better way to interpolate is to use some sort of spline. We introduce quaternions $\hat{\mathbf{a}}_i$ and $\hat{\mathbf{a}}_{i+1}$ between $\hat{\mathbf{q}}_i$ and $\hat{\mathbf{q}}_{i+1}$. Spherical cubic interpolation can be defined within the set of quaternions $\hat{\mathbf{q}}_i$, $\hat{\mathbf{a}}_i$, $\hat{\mathbf{a}}_{i+1}$, and $\hat{\mathbf{q}}_{i+1}$. Surprisingly, these extra quaternions are computed as shown below [404]^3:

$$\hat{\mathbf{a}}_i = \hat{\mathbf{q}}_i \exp\left[-\frac{\log(\hat{\mathbf{q}}_i^{-1}\hat{\mathbf{q}}_{i-1}) + \log(\hat{\mathbf{q}}_i^{-1}\hat{\mathbf{q}}_{i+1})}{4}\right]. \tag{4.54}$$

Spherical Quadrangle Interpolation = SQUAD

Since $\hat{\mathbf{q}}_i, \hat{\mathbf{q}}_{i+1}, \hat{\mathbf{a}}_i, \hat{\mathbf{a}}_{i+1}$ form a quadrangle

$$\begin{aligned}&\texttt{squad}(\hat{\mathbf{q}}_i, \hat{\mathbf{q}}_{i+1}, \hat{\mathbf{a}}_i, \hat{\mathbf{a}}_{i+1}, t) = \\ &\quad \texttt{slerp}(\texttt{slerp}(\hat{\mathbf{q}}_i, \hat{\mathbf{q}}_{i+1}, t), \texttt{slerp}(\hat{\mathbf{a}}_i, \hat{\mathbf{a}}_{i+1}, t), 2t(1-t))\end{aligned}$$

Interpolating Quaternions: The Hack

Interpolating quaternions should be done on the surface of a 3D unit sphere embedded in 4D space.

However, much simpler interpolation along a 4D straight line (open circles) followed by reprojection of the results onto the sphere (black circles) is often sufficient.