Part II: Visualizing Quaternion Geometry

Andrew J. Hanson

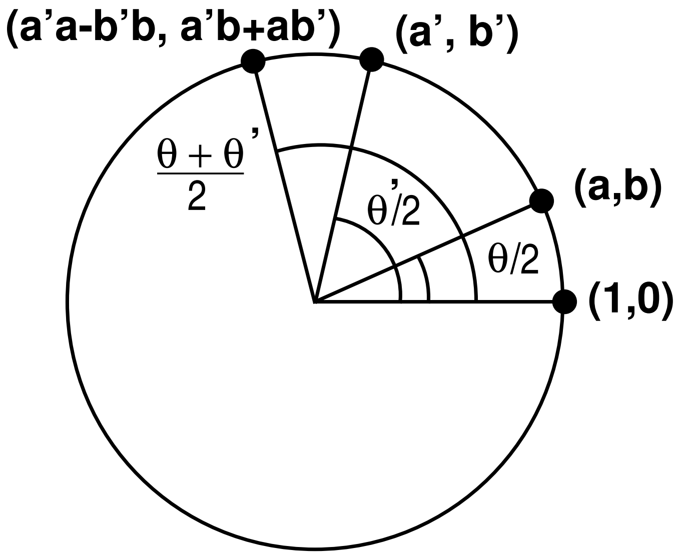
Indiana University

Part II: OUTLINE

- **The Spherical Projection Trick:** Visualizing unit vectors.
- **Quaternion Frames**
- **Quaternion Curves**
- **Quaternion Splines**

The Geometry of Quaternions

Recall (a, b) with $a^2 + b^2 = 1$ is a unit-length **complex number** or a **point on the unit circle** S^1 .

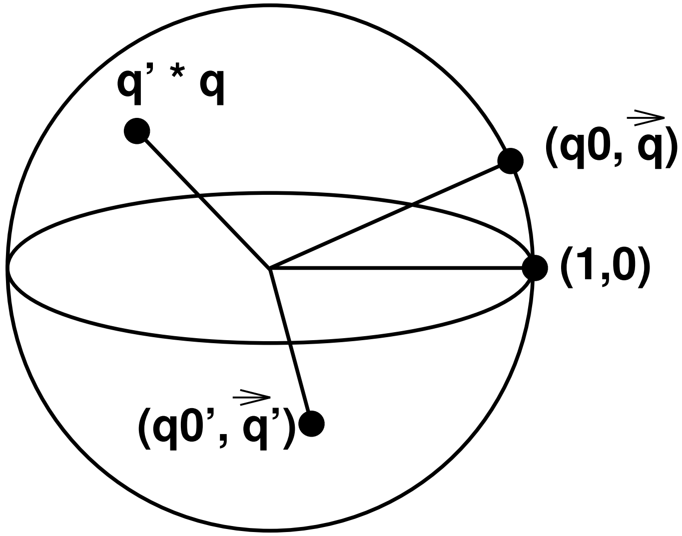


The Geometry of Quaternions . . .

Similarly, $q = (q_0, \vec{\mathrm{q}})$ with ${q_0}^2 + \vec{\mathrm{q}}^2 = 1$ is a unit-length **quaternion** or a **point on the unit 3-sphere** S^3.

The Geometry of Quaternions . . .

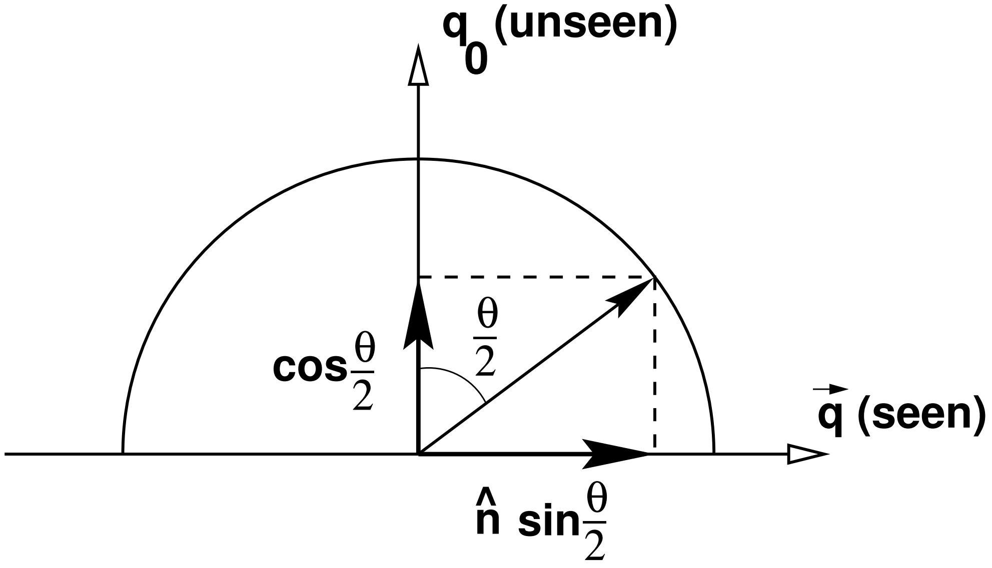
Rotations combine by taking the quaternion product of the *geometric* values of 4D points on S^3:



Visualizing a Quaternion??

Learn how to *Visualize* a quaternion by starting with a visualization of a point on S^1, the circle:

$$q_0 = \sqrt{1 - (q_1)^2}$$

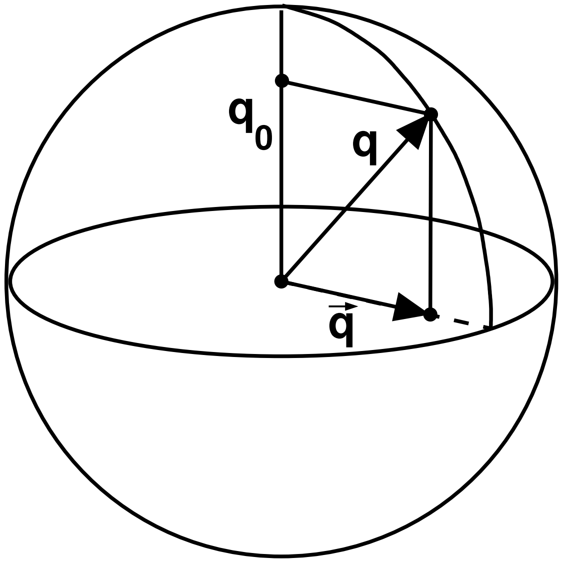


Demo: 2D Unit Vector Projection

Visualizing a Quaternion?? . . .

Next, visualize a point on S^2, the ordinary sphere using only the projection $\vec{\mathrm{q}}$:

$$q_0 = \sqrt{1 - (q_1)^2 - (q_2)^2}$$



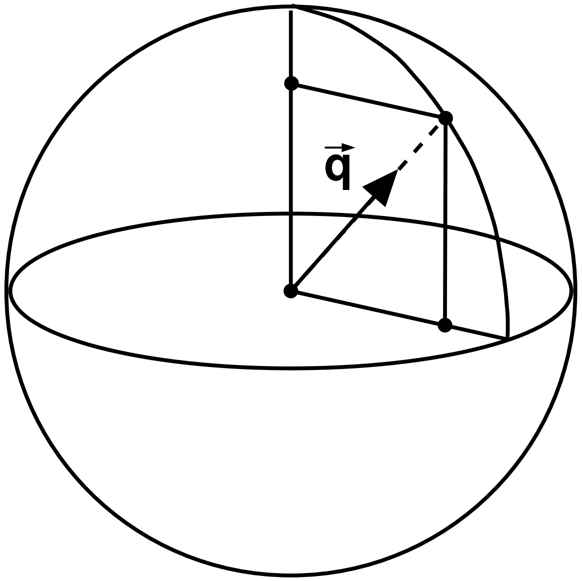
Demo: 3D Unit Vector Projection

Visualizing a Quaternion?? . . .

Finally, visualize a point on S^3, the quaternion space:

DISPLAY only $\vec{\mathrm{q}}$, but **INFER**

$$q_0 = \sqrt{1 - (q_1)^2 - (q_2)^2 - (q_3)^2}$$



Demo: 4D Unit Vector Projection

Visualize Quaternion Rotations

Each 4D quaternion point $q = (\cos\frac{\theta}{2}, \hat{\mathbf{n}}\sin\frac{\theta}{2})$ is a frame — a 3×3 rotation matrix generated by applying $R(\theta, \hat{\mathbf{n}})$ to the identity frame.

Identity Matrix is the quaternion $q = (1, 0, 0, 0)$.

Visualize q using only the VECTOR part $\vec{\mathbf{q}}$, so Identity is the zero vector.

Visualize Quaternion Rotations . . .

The quaternion rotation by θ about $\hat{\mathbf{n}}$:

$$q = (q_0,\, \mathbf{q}) = (\cos(\theta/2),\, \hat{\mathbf{n}}\sin(\theta/2))$$

represents the matrix $R(\theta, \hat{\mathbf{n}})$.

Action of rotating Identity by θ about $\hat{\mathbf{n}}$:

$q * (1, 0, 0, 0)$ gives Vector part:

$$\vec{\mathbf{0}} \Rightarrow \hat{\mathbf{n}}\sin(\theta/2)$$

Quaternion Interpretation

Rotation axis is Fundamental 3-vector: We know

$$q_0 = \cos(\theta/2), \quad \vec{\mathbf{q}} = \hat{\mathbf{n}} \sin(\theta/2)$$

We also know that *any coordinate frame* M can be written as $M = R(\theta, \hat{\mathbf{n}})$.

Therefore

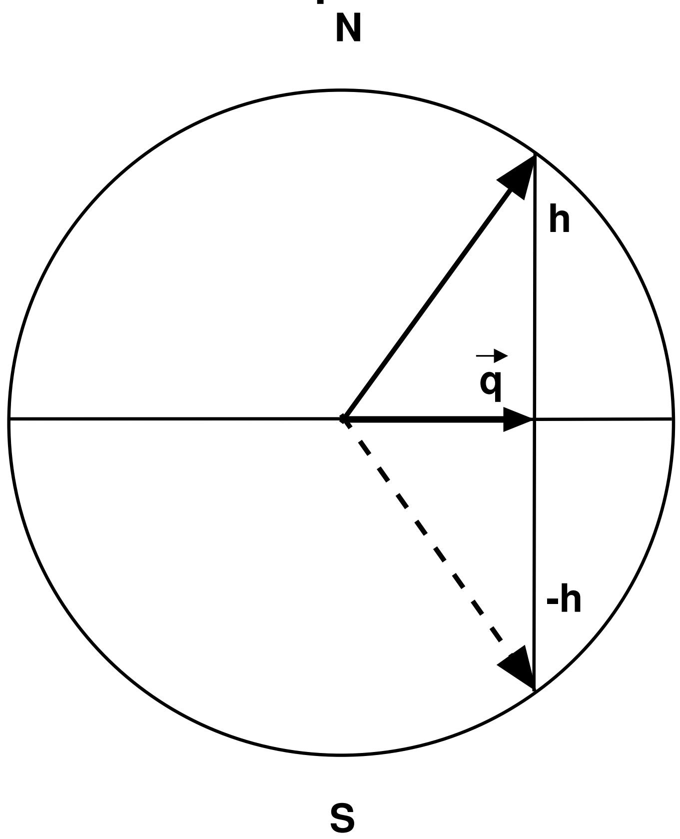
$\boxed{\text{}}$ **$\vec{\mathrm{q}}$ points exactly along the axis we have to rotate around to go from identity I to M, and the length of $\vec{\mathrm{q}}$ tells us how much to rotate.**

Demo: Growing Quaternion = Rotation about Axis

$\widehat{\mathrm{q}}$ = required Euler theorem axis.

Displaying spherical points

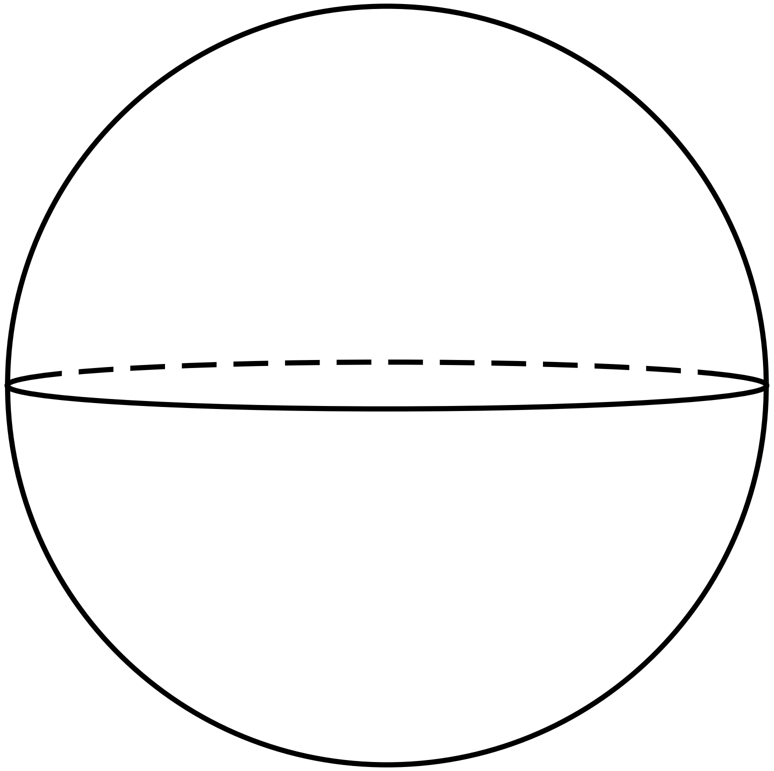
Displaying a point on a sphere is ambiguous:



The same horizontal projection is shared by the North vector $(h, \vec{\mathrm{q}})$ and the South vector $(-h, \vec{\mathrm{q}})$.

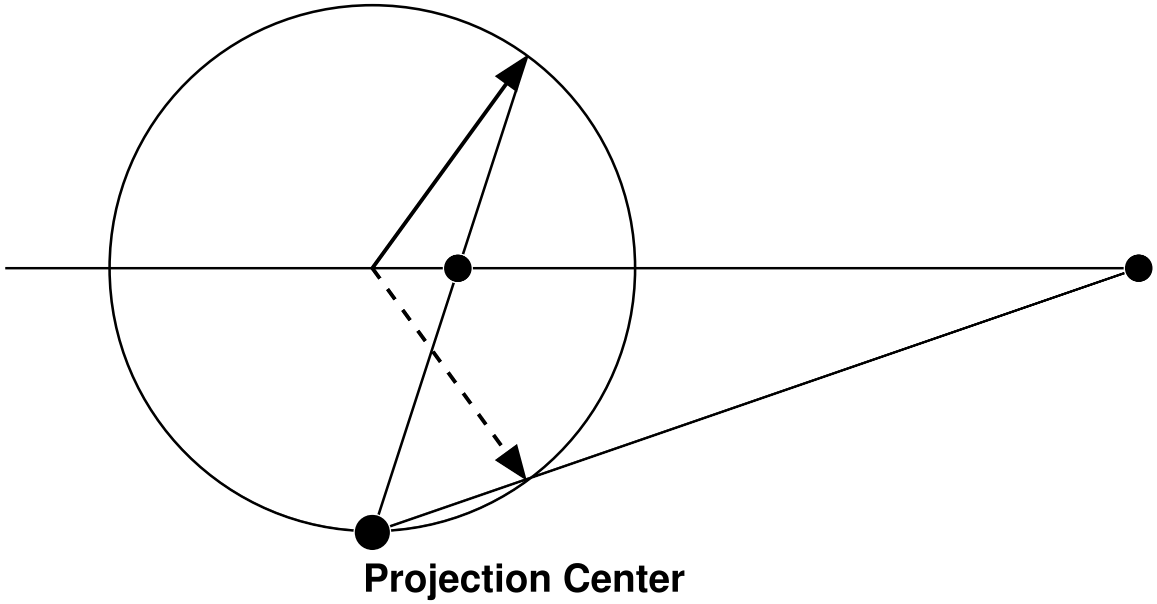
Orbit is Front/Back Line

2D projection of orbit around sphere has two half circles that line up in 2D, match at two edge points only in 3D.



Projective spherical points

Polar Projection removes ambiguity, destroys symmetry:



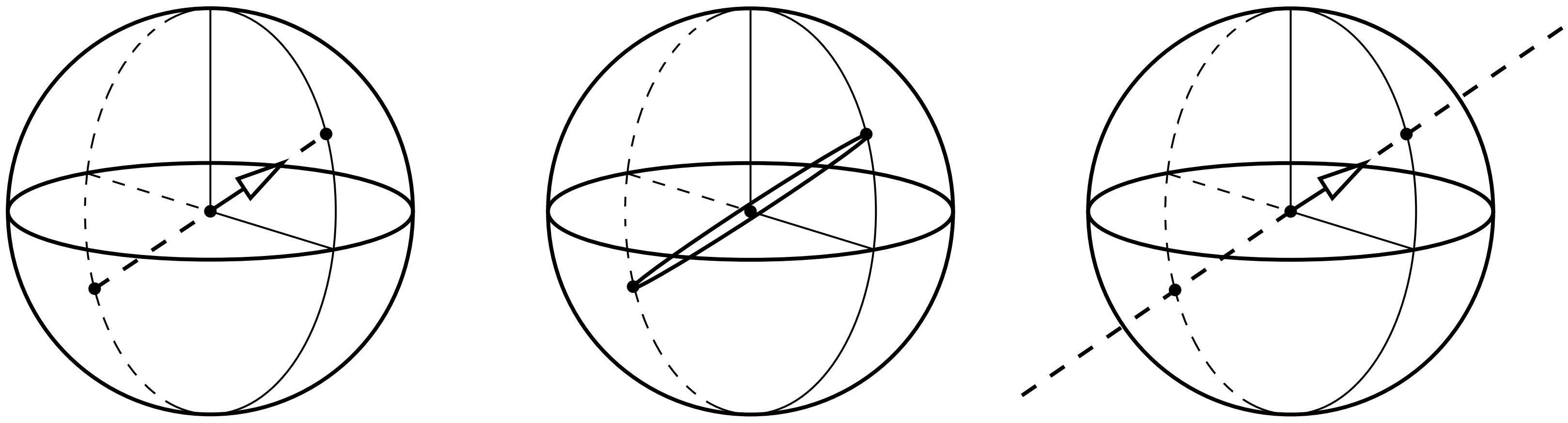
the North vector $(h, \vec{\mathrm{q}})$ and the South vector $(-h, \vec{\mathrm{q}})$ project to *different points* on the line.

Displaying S^3

A quaternion point can be displayed in

- **Parallel Projection:** so $q = (h, \vec{\mathrm{q}})$ lines up with $q = (-h, \vec{\mathrm{q}})$,
- **Polar projection:** so only the "north pole" projects within the unit sphere, and "south pole" is at ∞ of R^3.

Displaying S^3 ...



(a) (b) (c)

(a) Usual vector quaternion point. (b) Orbits through northern and southern hemispheres. (c) Polar projection: north pole at origin, south pole at infinity.

Quaternion Curves

Represent Families of Frames

Each orientation is a *4D point* on the 3-sphere representing a quaternion.

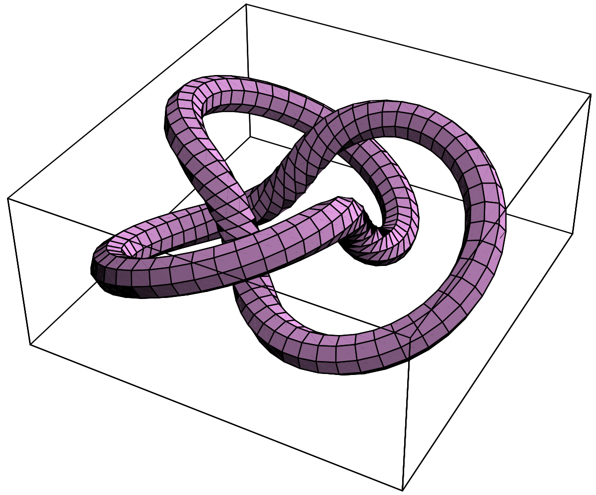
Thus families of frames, which are really rotation matrices, become **curves** on the 3-sphere.

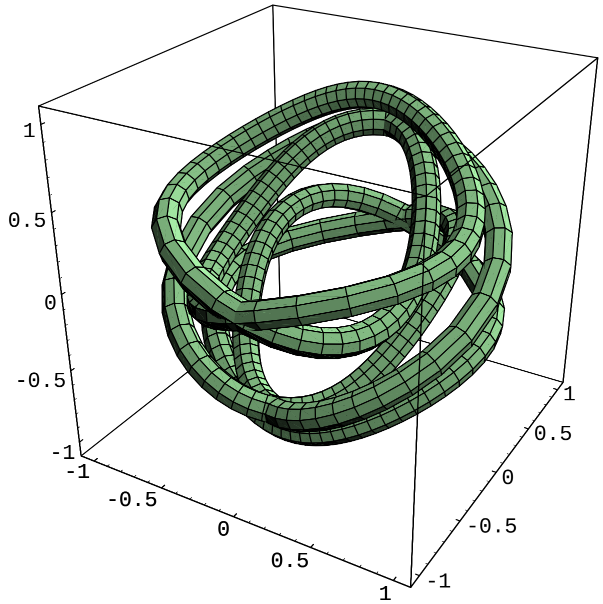
$\Rightarrow$ treat these curves just like any other curve...

Demo: Simple Quaternion Curves

Families of Quaternion Frames, . . .

Example: torus knot and its (twice around) quaternion Frenet frame:

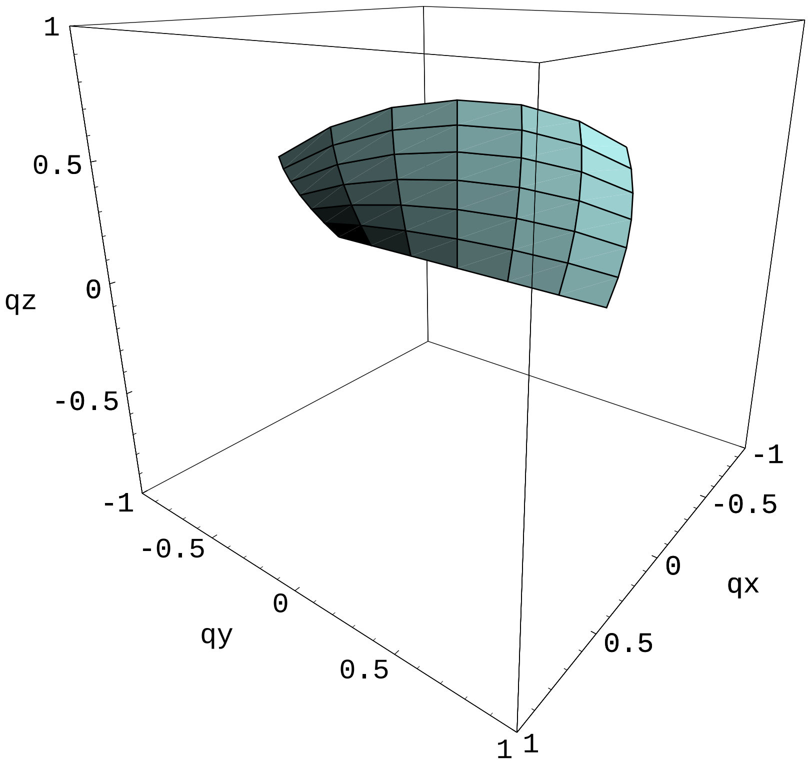


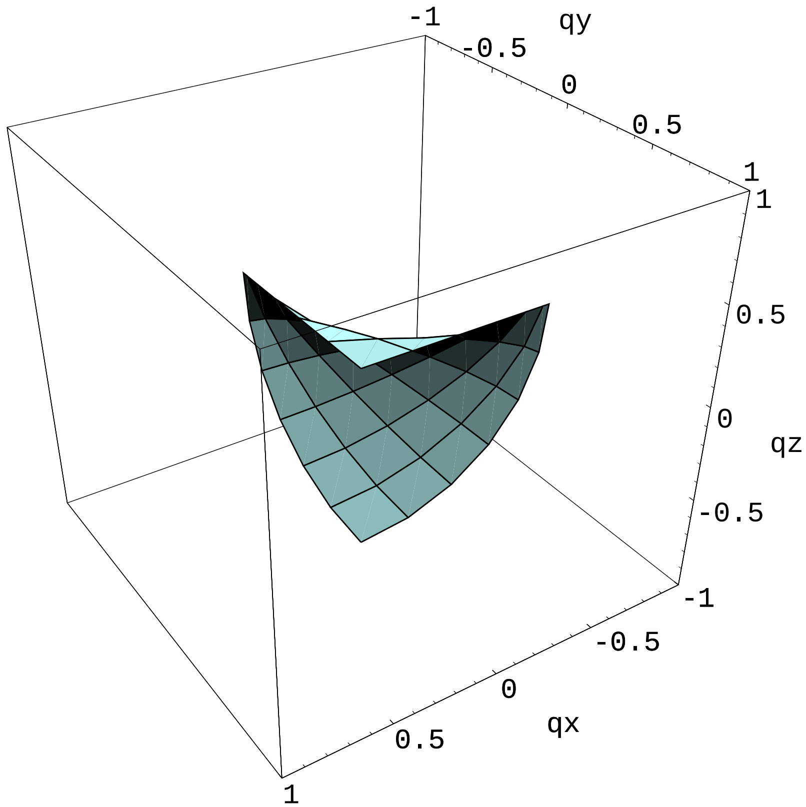


see: Hanson and Ma, “Quaternion Frame Approach to Streamline Visualization,” *IEEE Trans. on Visualiz. and Comp. Graphics*, **1**, No. 2, pp. 164–174 (June, 1995).

Families of Quaternion Frames, . . .

Example: **Surface** describes two-degree-of-freedom joint





Remark: Group theory *demands* that x and y rotations combine to give some z rotation!

Quaternion Interpolations

- Shoemake (Siggraph '85) proposed using quaternions instead of Euler angles to get smooth frame interpolations without **Gimbal Lock**:

  BEST CHOICE: Animate objects and cameras using rotations represented on S^3 by quaternions

Interpolating on Spheres

N-dimensional spherical interpolation employs the "SLERP," a constant angular velocity transition between two directions, $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$:

$$\boxed{\begin{aligned} \hat{\mathbf{n}}_{12}(t) &= \text{Slerp}(\hat{\mathbf{n}}_1, \hat{\mathbf{n}}_2, t) \\ &= \hat{\mathbf{n}}_1 \frac{\sin((1-t)\theta)}{\sin(\theta)} + \hat{\mathbf{n}}_2 \frac{\sin(t\theta)}{\sin(\theta)} \end{aligned}}$$

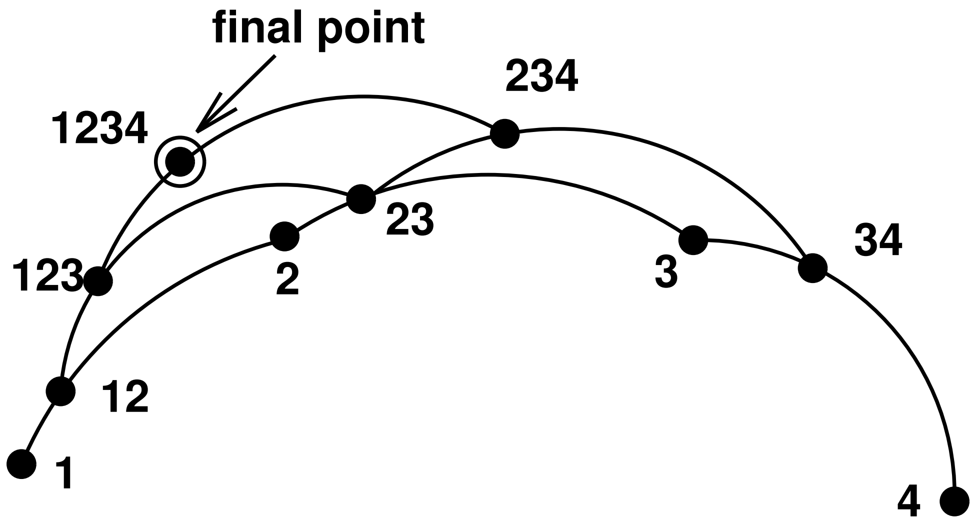
where $\cos\theta = \hat{\mathbf{n}}_1 \cdot \hat{\mathbf{n}}_2$.

Remark on Spherical Interpolation

(The SLERP formula is simply the result of applying a **Gram-Schmidt decomposition** to the two vectors while enforcing unit norm in *any dimension*.)

Quaternion Interpolations

Many variations have been proposed since then; simplest is simply to apply the formula iteratively to give analog of the de Casteljau spline construction:



Spline Families

Schlag (in Graphics Gems II (1991)) gives recursive form for several cubic splines:

$$
\begin{aligned}
&S(x_1, x_2, x_3, x_4, t) = \\
&L(L(L(x_1, x_2, f_{12}(t)),\; L(x_2, x_3, f_{23}(t)),\; f_{123}(t)), \\
&\quad L(L(x_2, x_3, f_{23}(t)),\; L(x_3, x_4, f_{34}(t)),\; f_{234}(t)), \\
&\qquad f(t))
\end{aligned}
$$

Spline Families . . .

For **Euclidean space**, the interpolator is

$$L(a, b, t) = a(1 - t) + bt$$

while for **Spherical space**, the interpolator is

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

where $a \cdot b = \cos\theta$.

Spline Families . . .

Catmull-Rom

$$f_{12} = t + 1 \qquad f_{23} = t \qquad f_{34} = t - 1$$

$$f_{123} = \frac{(t+1)}{2} \qquad f_{234} = \frac{t}{2}$$

$$f = t$$

Spline Families . . .

Bezier

$$\boxed{\begin{array}{ccccc} f_{12} = t & & f_{23} = t & & f_{34} = t \\ & f_{123} = t & & f_{234} = t & \\ & & f = t & & \end{array}}$$

Spline Families . . .

Uniform B-spline

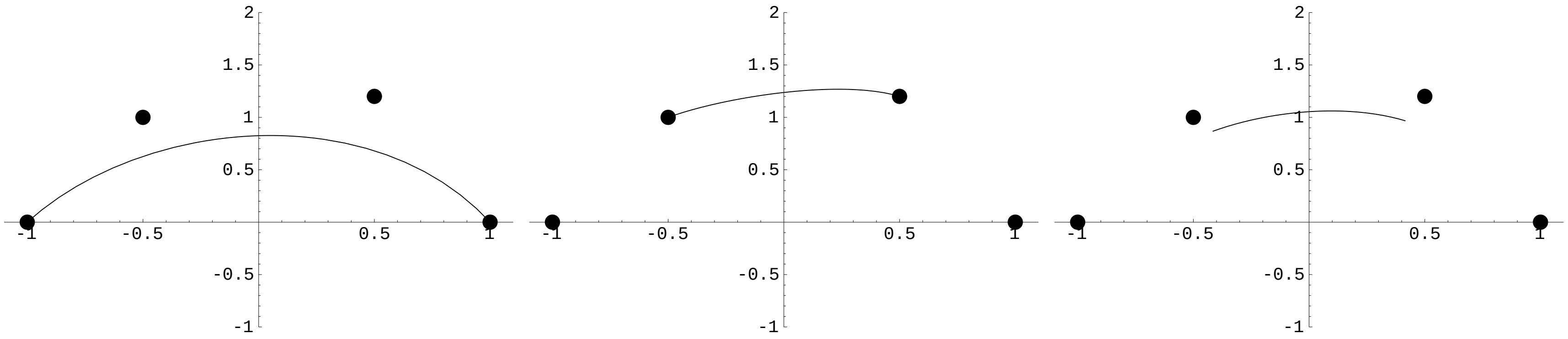
$$f_{12} = \frac{(t+2)}{3} \qquad f_{23} = \frac{(t+1)}{3} \qquad f_{34} = \frac{t}{3}$$

$$f_{123} = \frac{(t+1)}{2} \qquad f_{234} = \frac{t}{2}$$

$$f = t$$

Plane Interpolations

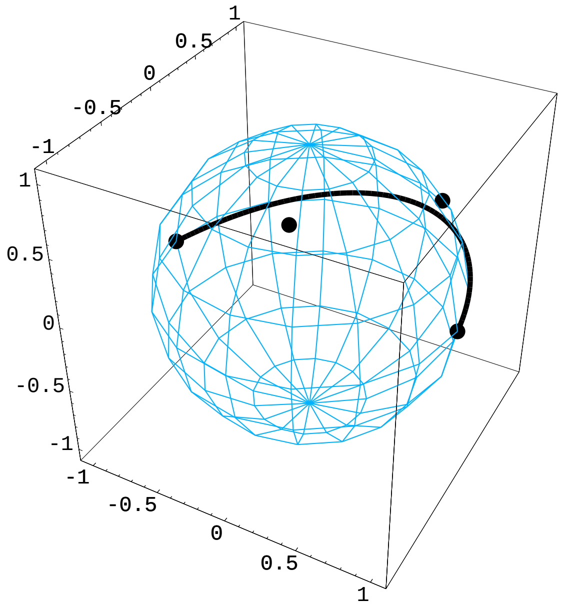
In Euclidean space, these three basic cubic splines look like this:



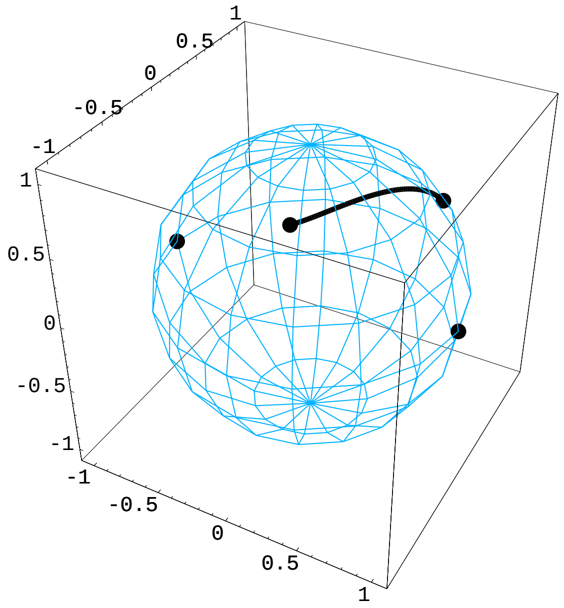
Bezier Catmull-Rom Uniform B

The differences are in the derivatives: Bezier has to start matching all over at every fourth point; Catmull-Rom matches the first derivative; and B-spline is the cadillac, matching all derivatives but no *control points*.

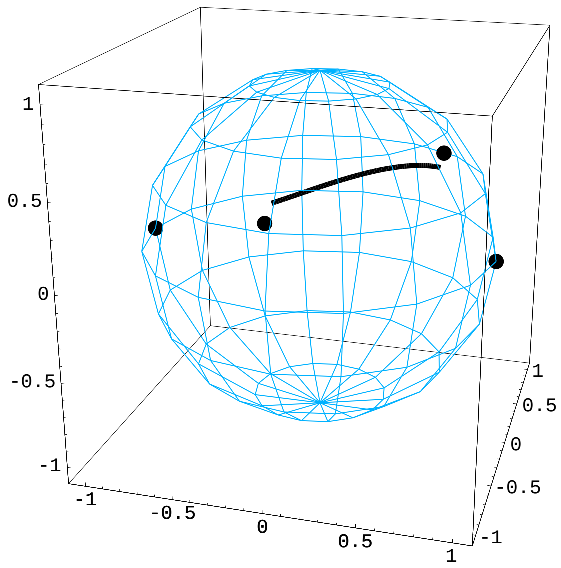
Spherical Interpolations



Bezier

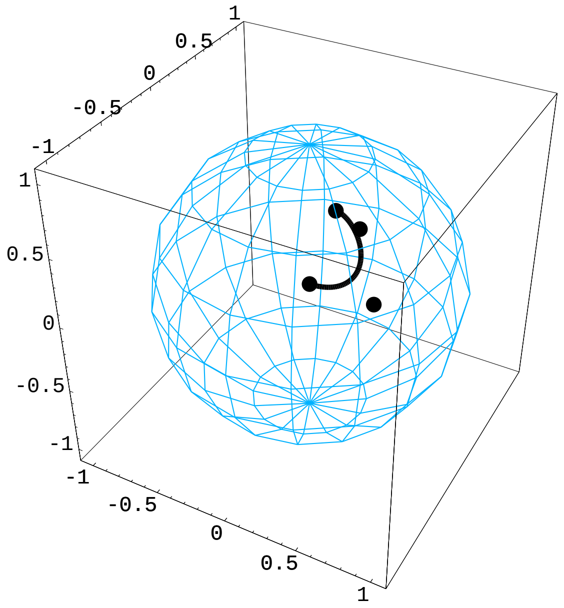


Catmull-Rom

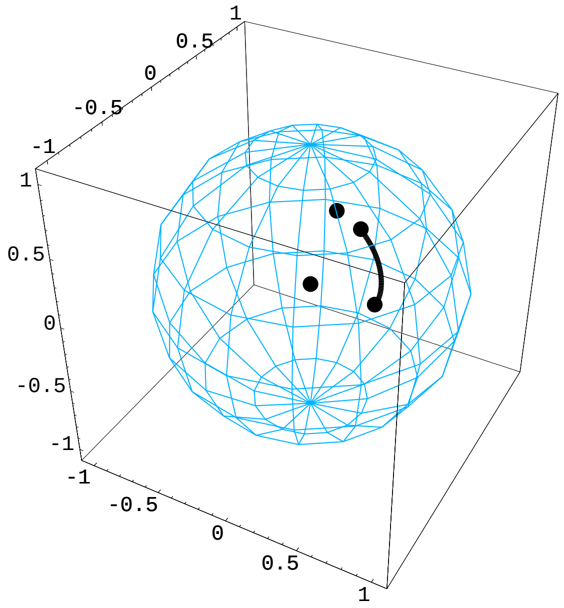


Uniform B

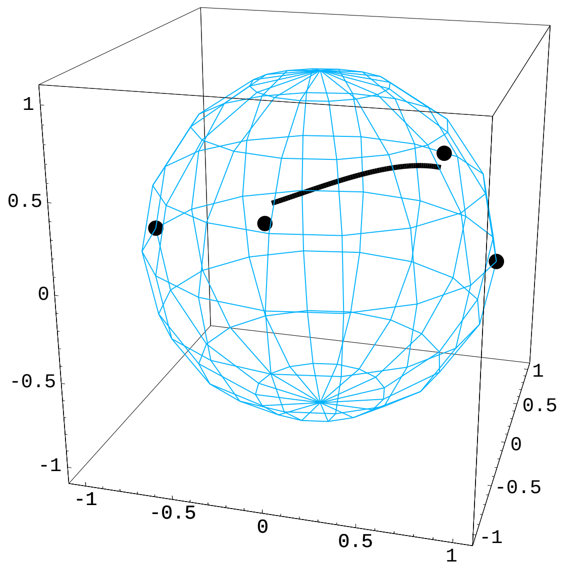
Quaternion Interpolations



Bezier



Catmull-Rom



Uniform B

Demo: Interpolated Keyframe Quaternion Curves

Optional:

Exponential Quaternion Map

Exponential map (Kim, Kim, and Shin; see also Grassia) has these advantages:

- **Derivative Computation Simpler.** Initial velocity conditions work.
- **Derivative Matching Simpler.** Easier to match neighboring derivatives.
- **No Renormalization Req'd.**
  4 vars, 3 DOF $\Rightarrow$ 3 independent vars, constraints automatic.

Exponential Quaternion Map . . .

METHOD: use only three "vector" degrees of freedom $\vec{\mathbf{n}} = \hat{\mathbf{n}}\,\theta/2$: exponentiating gives four-component quaternion obeying unit 3-sphere constraint:

$$\begin{aligned} q(\vec{\mathbf{n}}) &= e^{(\mathbf{I}\cdot\hat{\mathbf{n}}\,\theta/2)} \\ &= (\cos(\theta/2),\, \hat{\mathbf{n}}\sin(\theta/2)) \end{aligned}$$

Note (Grassia): the quaternion frame evolution equations can be then be written *directly* in terms of the 3-vector $\vec{\mathbf{n}}$!

Exponential Quaternion Map . . .

Then, using *logarithms* of quaternion control points, e.g.,

$$\vec{\mathbf{n}} = \log q = \left(\frac{\theta}{2}\right)\frac{\mathbf{q}}{|\mathbf{q}|} = \cos^{-1}(q_0)\frac{\mathbf{q}}{|\mathbf{q}|}$$

where $\vec{\mathbf{n}}$ is *not* normalized, we can use splines in the reduced *Euclidean* 3D space.

Exponential Quaternion Map . . .

The trick:

Wherever you would see $x_0 + t(x_1 - x_0)$ in a Euclidean spline, replace $(x_1 - x_0)$ by

$$\omega = \log((q_0)^{-1} * q_1)$$

(This is easy: quaternion multiplication gives a new $Q = (Q_0, \mathbf{Q})$, and $\omega = \log Q$ is computed as usual.)

Exponential Quaternion Map . . .

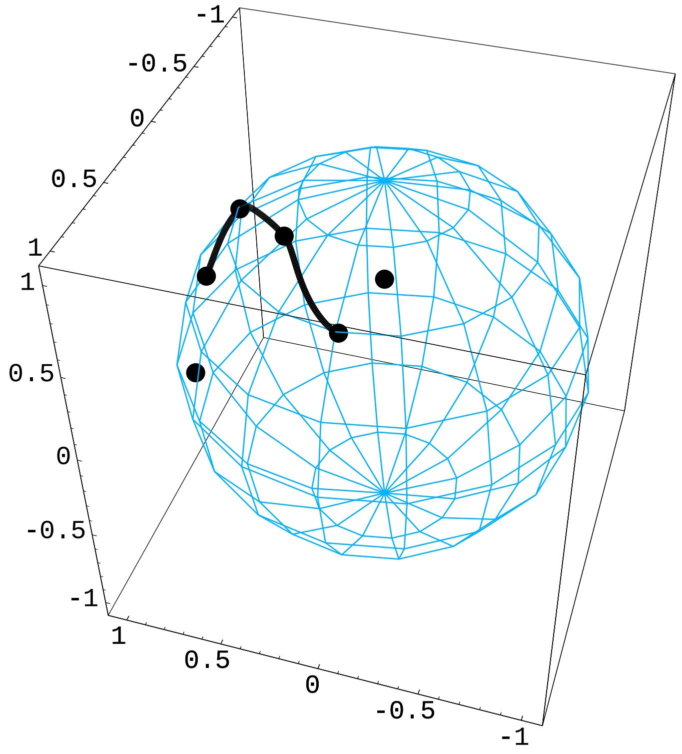
Then, with $\omega_i = \mathsf{log}((q_{i-1})^{-1} * q_i)$, let

$$q(t) = q_0 \prod_{i=1}^{k} \mathsf{exp}(\omega_i \alpha_i(t))$$

$\Rightarrow$ Euclidean spline $\alpha_i(t)$ methods can be used, and derivative computations are simplified.

Exponential Quaternion Map . . .

Example of a many-control-point Catmull-Rom spline done using the exponential method:



Even More Quaternion Interpolations . . .

A number of other approaches can be found in the bibliography. Other literature includes:

- **Barr et al.** Global optimization emulating vanishing 4th derivative of Euclidean cubic splines, but on quaternion three-sphere.

- **Jüttler et al.:** Generalized rational splines, including quaternion component as well as a spatial component.

SUMMARY

- **The Spherical Projection Trick:** Visualizing unit vectors.
- **Quaternion Frames:** $\hat{\mathbf{n}}$ in quaternion tells how to make frame.
- **Quaternion Curves:** are like any other curve; spline applications.