CSCI 520 Computer Animation and Simulation

Quaternions and Rotations

Jernej Barbic
University of Southern California

Rotations

- Very important in computer animation and robotics
- Joint angles, rigid body orientations, camera parameters
- 2D or 3D

Rotations in Three Dimensions

- Orthogonal matrices:

$$RR^{\mathrm{T}} = R^{\mathrm{T}}R = I$$
$$\det(R) = 1$$

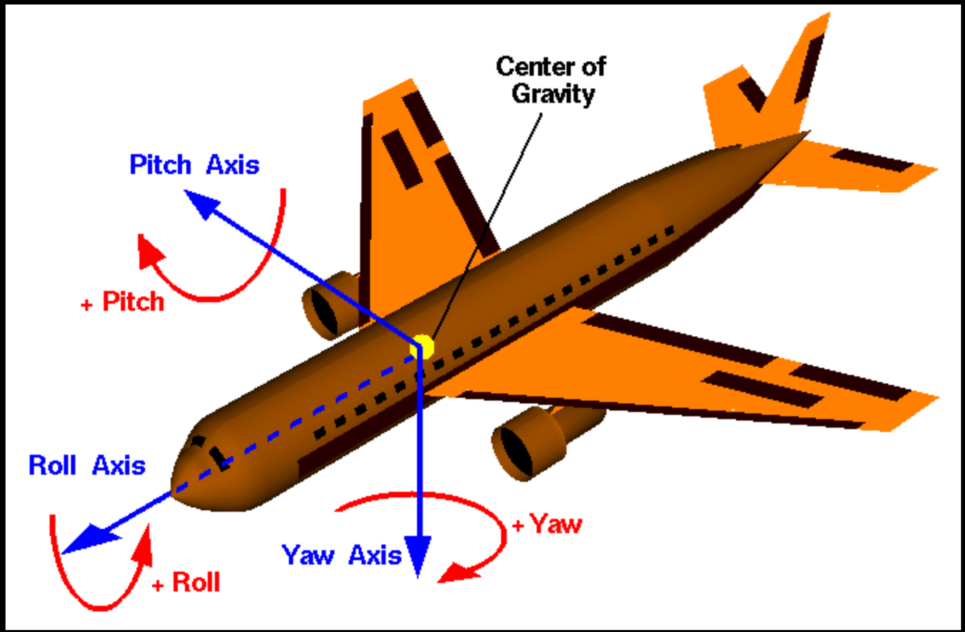
$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

Representing Rotations in 3D

- Rotations in 3D have essentially three parameters
- Axis + angle (2 DOFs + 1DOFs)
  - How to represent the axis?
    Longitude / lattitude have singularities
- 3x3 matrix
  - 9 entries (redundant)

Representing Rotations in 3D

- Euler angles
  - roll, pitch, yaw
  - no redundancy (good)
  - gimbal lock singularities



Source: Wikipedia

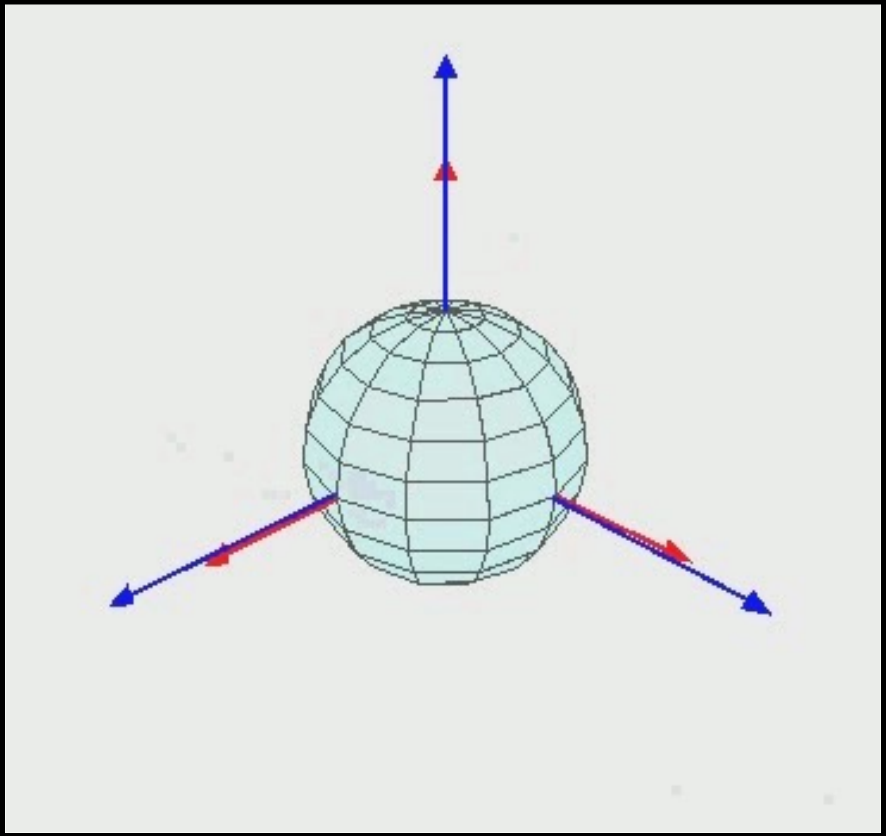
- Quaternions
  - generally considered the “best” representation
  - redundant (4 values), but only by one DOF (not severe)
  - stable interpolations of rotations possible

Euler Angles

1. Yaw
   rotate around y-axis

2. Pitch
   rotate around (rotated) x-axis

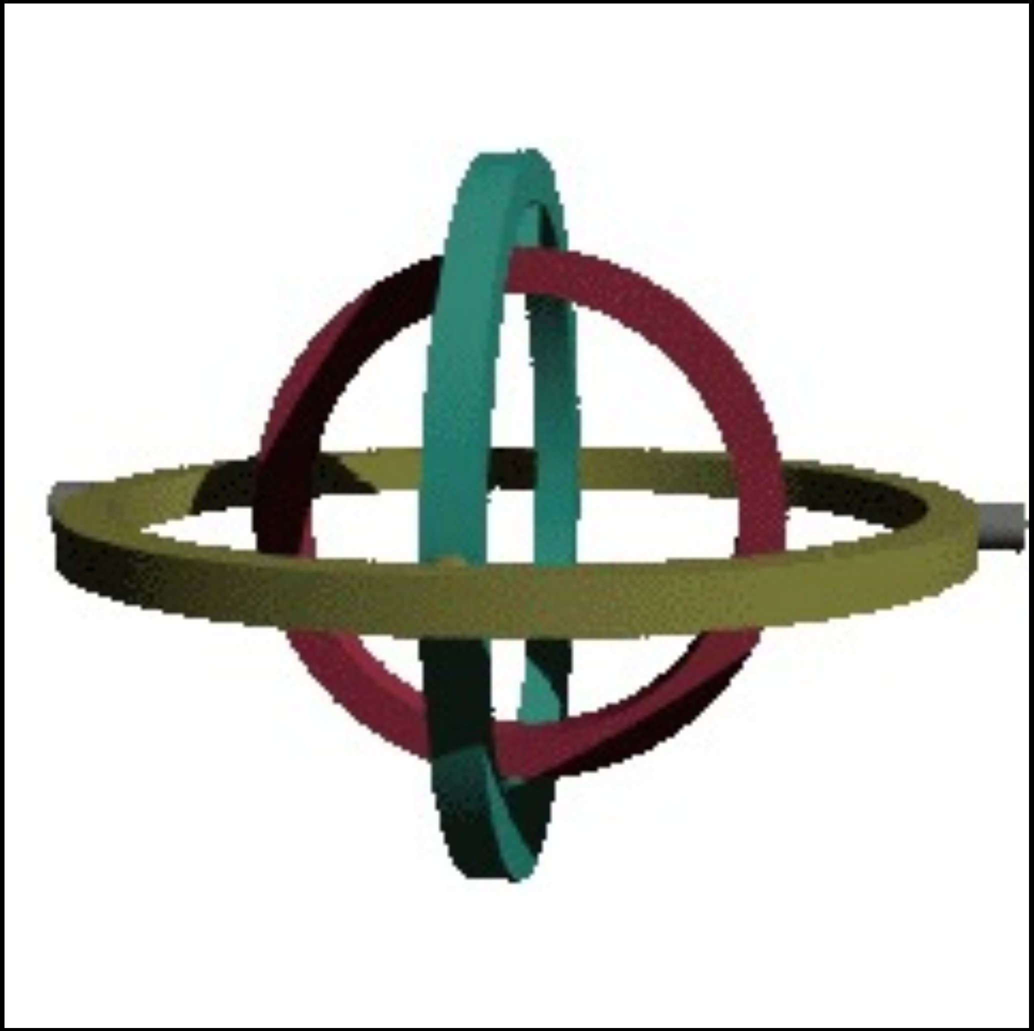
3. Roll
   rotate around (rotated) y-axis



Source: Wikipedia

Gimbal Lock

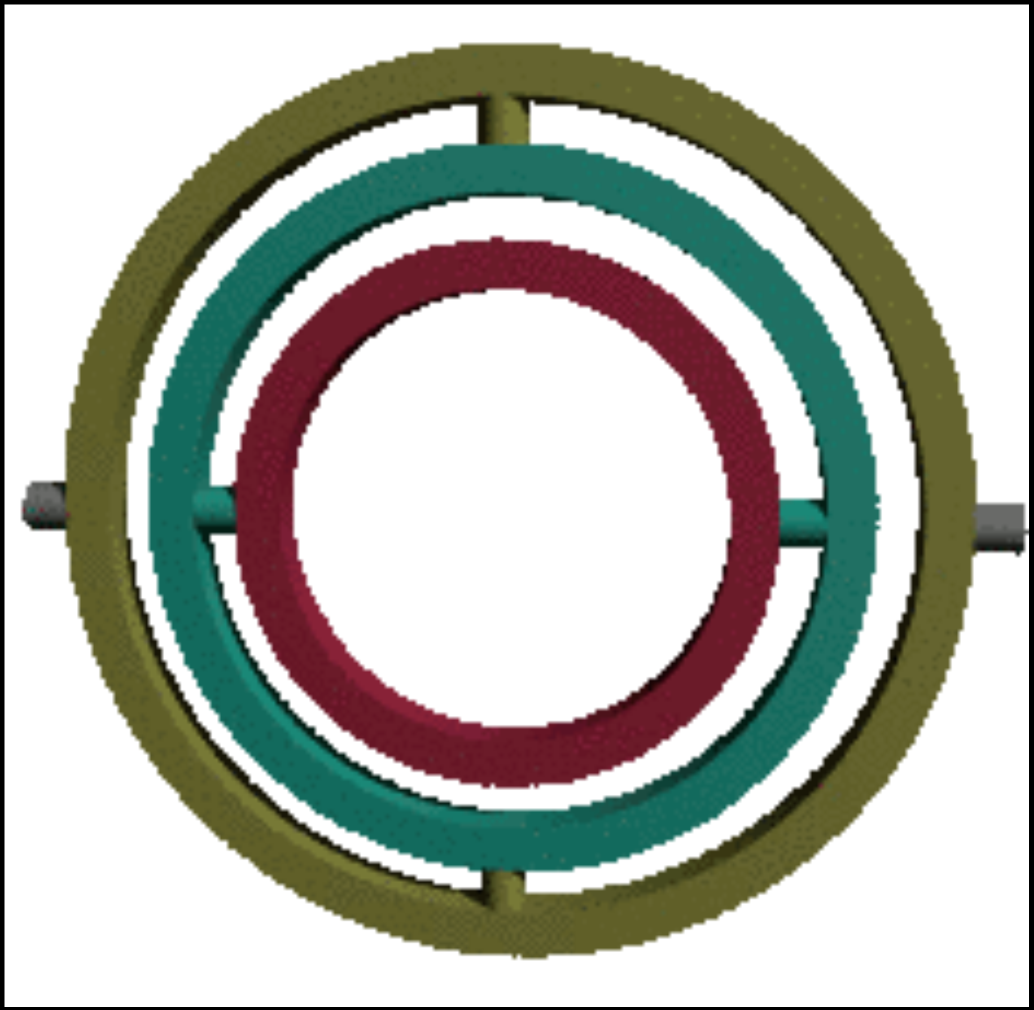
When all three gimbals are lined up (in the same plane), the system can only move in two dimensions from this configuration, not three, and is in *gimbal lock*.



Source: Wikipedia

Gimbal Lock

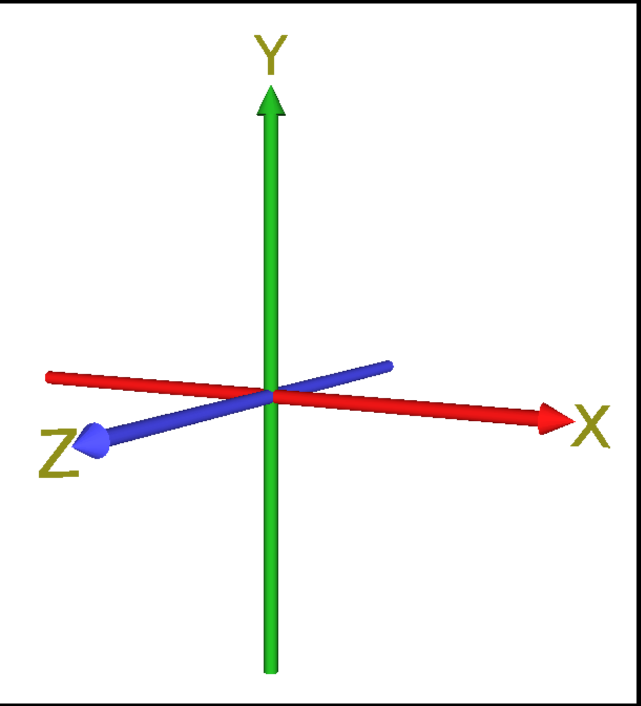
When all three gimbals are lined up (in the same plane), the system can only move in two dimensions from this configuration, not three, and is in *gimbal lock*.



Source: Wikipedia

Choice of rotation axis sequence for Euler Angles

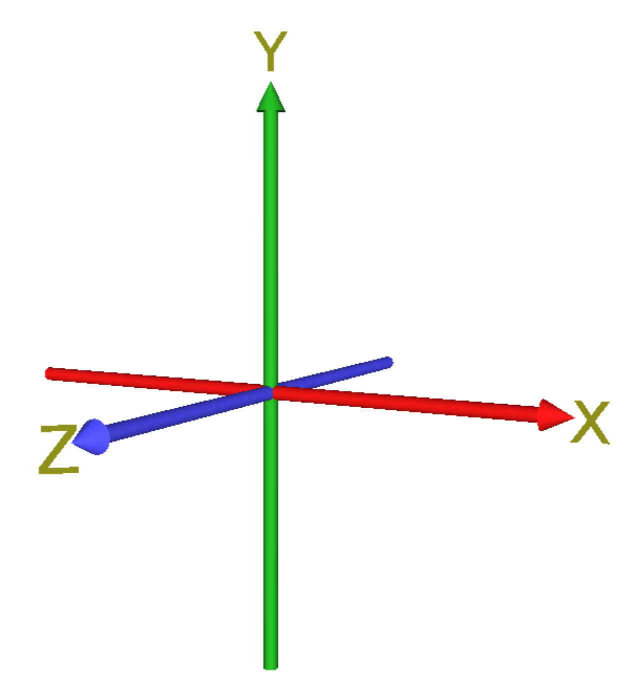
- 12 choices:
  XYX, XYZ, XZX, XZY,
  YXY, YXZ, YZX, YZY,
  ZXY, ZXZ, ZYX, ZYZ



- Each choice can use static axes, or rotated axes, so we have a total of 24 Euler Angle versions!

Example: XYZ Euler Angles

- First rotate around X by angle θ_1, then around Y by angle θ_2, then around Z by angle θ_3 .



- Used in CMU Motion Capture Database AMC files

- Rotation matrix is:

$$R = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 \\ \sin(\theta_3) & \cos(\theta_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_2) & 0 & \sin(\theta_2) \\ 0 & 1 & 0 \\ -\sin(\theta_2) & 0 & \cos(\theta_2) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_1) & -\sin(\theta_1) \\ 0 & \sin(\theta_1) & \cos(\theta_1) \end{bmatrix}$$

Outline

- Rotations
- Quaternions
- Quaternion Interpolation

Quaternions

- Generalization of complex numbers

- Three imaginary numbers: $\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{k}$

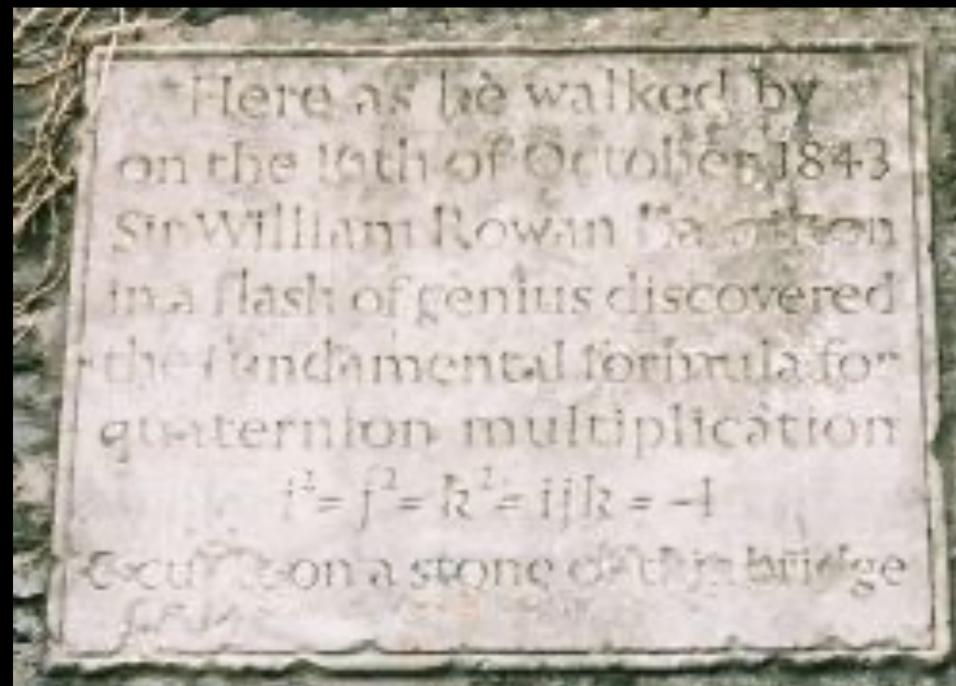
  $\boldsymbol{i}^2 = -1$, $\boldsymbol{j}^2 = -1$, $\boldsymbol{k}^2 = -1$,
  $\boldsymbol{ij} = \boldsymbol{k}$, $\boldsymbol{jk} = \boldsymbol{i}$, $\boldsymbol{ki} = \boldsymbol{j}$, $\boldsymbol{ji} = -\boldsymbol{k}$, $\boldsymbol{kj} = -\boldsymbol{i}$, $\boldsymbol{ik} = -\boldsymbol{j}$

- $q = s + x\,\boldsymbol{i} + y\,\boldsymbol{j} + z\,\boldsymbol{k}$, s,x,y,z are scalars

Quaternions

- Invented by Hamilton in 1843 in Dublin, Ireland

- 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
  $i^2 = j^2 = k^2 = i\,j\,k = -1$
  & cut it on a stone of this bridge.



Source: Wikipedia

Quaternions

- Quaternions are **not** commutative!

  $q_1 \, q_2 \neq q_2 \, q_1$

- However, the following hold:

  $(q_1 \, q_2) \, q_3 = q_1 \, (q_2 \, q_3)$
  $(q_1 + q_2) \, q_3 = q_1 \, q_3 + q_2 \, q_3$
  $q_1 \, (q_2 + q_3) = q_1 \, q_2 + q_1 \, q_3$
  $\alpha \, (q_1 + q_2) = \alpha \, q_1 + \alpha \, q_2$ (α is scalar)
  $(\alpha q_1) \, q_2 = \alpha \, (q_1 q_2) = q_1 \, (\alpha q_2)$ (α is scalar)

- I.e., all usual manipulations are valid, except cannot reverse multiplication order.

Quaternions

- Exercise: multiply two quaternions

$(2 - \boldsymbol{i} + \boldsymbol{j} + 3\boldsymbol{k})\,(-1 + \boldsymbol{i} + 4\boldsymbol{j} - 2\boldsymbol{k}) = \ldots$

Quaternion Properties

- $q = s + x\,\boldsymbol{i} + y\,\boldsymbol{j} + z\,\boldsymbol{k}$
- Norm: $|q|^2 = s^2 + x^2 + y^2 + z^2$
- Conjugate quaternion: $\bar{q} = s - x\,\boldsymbol{i} - y\,\boldsymbol{j} - z\,\boldsymbol{k}$
- Inverse quaternion: $q^{-1} = \bar{q} \,/\, |q|^2$
- Unit quaternion: $|q| = 1$
- Inverse of unit quaternion: $q^{-1} = \bar{q}$

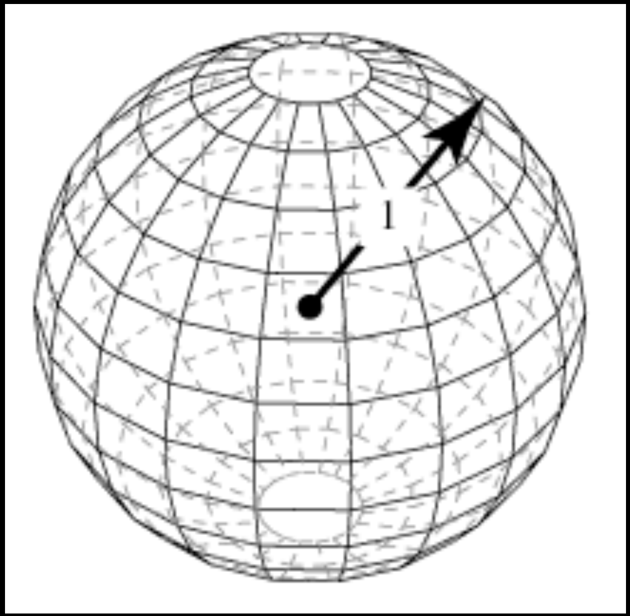
Quaternions and Rotations

- Rotations are represented by ***unit*** quaternions

- $q = s + x\,\boldsymbol{i} + y\,\boldsymbol{j} + z\,\boldsymbol{k}$

  $s^2 + x^2 + y^2 + z^2 = 1$

- Unit quaternion sphere (unit sphere in 4D)



Source:
Wolfram Research

unit sphere
in 4D

Rotations to Unit Quaternions

- Let (unit) rotation axis be $[u_x, u_y, u_z]$, and angle θ
- Corresponding quaternion is

  $q = \cos(\theta/2) +$
  $\quad \sin(\theta/2)\, u_x\, \boldsymbol{i} + \sin(\theta/2)\, u_y\, \boldsymbol{j} + \sin(\theta/2)\, u_z\, \boldsymbol{k}$

- Composition of rotations q_1 and q_2 equals $q = q_2\, q_1$
- 3D rotations do not commute!

Unit Quaternions to Rotations

- Let v be a (3-dim) vector and let q be a unit quaternion
- Then, the corresponding rotation transforms vector v to q ***v*** q^{-1}

  (***v*** is a quaternion with scalar part equaling 0, and vector part equaling v)

For q = a + b ***i*** + c ***j*** + d ***k***

$$R = \begin{pmatrix} a^2+b^2-c^2-d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & a^2-b^2+c^2-d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & a^2-b^2-c^2+d^2 \end{pmatrix}$$

Quaternions

- Quaternions q and -q give the same rotation!

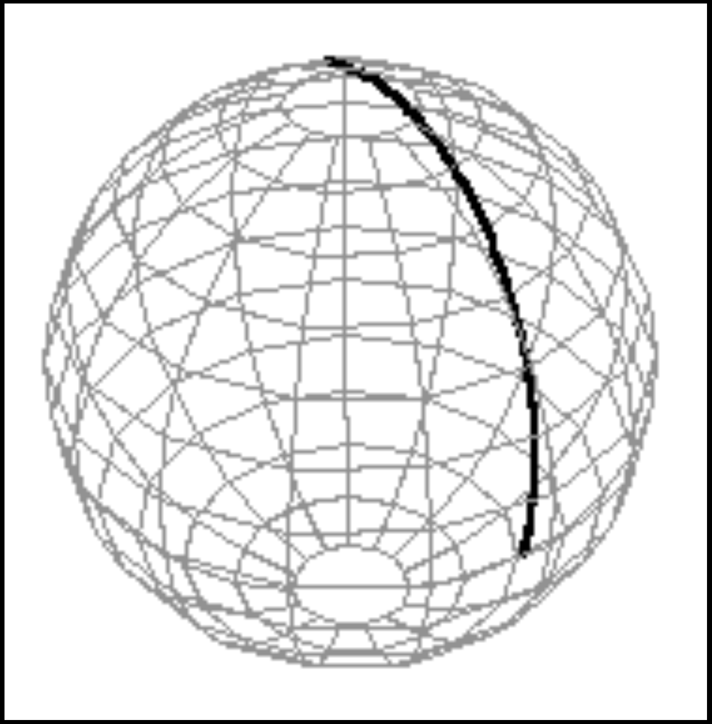
- Other than this, the relationship between rotations and quaternions is unique

Outline

- Rotations
- Quaternions
- Quaternion Interpolation

Quaternion Interpolation

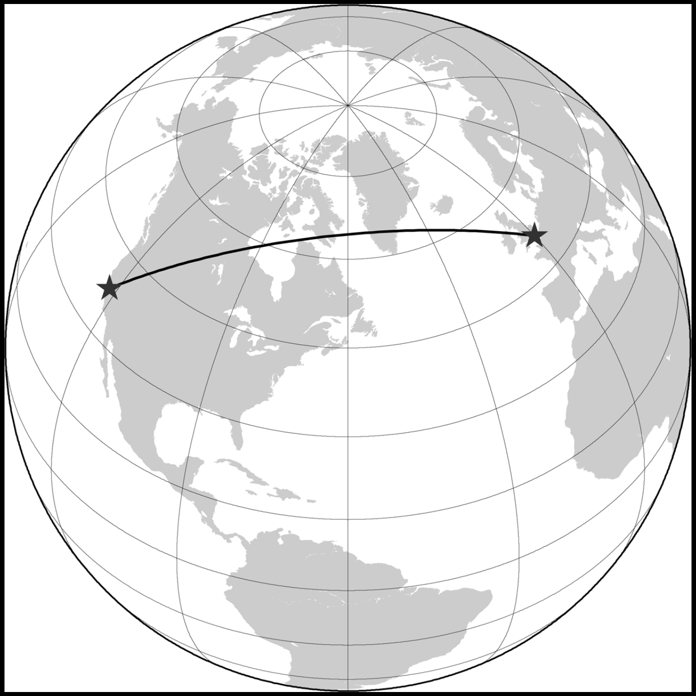
- Better results than Euler angles
- A quaternion is a point on the 4-D unit sphere
- Interpolating rotations corresponds to curves on the 4-D sphere



Source:
Wolfram Research

Spherical Linear intERPolation (SLERPing)

- Interpolate along the great circle on the 4-D unit sphere



- Move with constant angular velocity along the great circle between the two points

San Francisco
to London

- Any rotation is given by two quaternions, so there are two SLERP choices; pick the shortest

SLERP

$$\mathrm{Slerp}(q_1, q_2, u) = \frac{\sin((1-u)\theta)}{\sin(\theta)} q_1 + \frac{\sin(u\theta)}{\sin(\theta)} q_2$$

$$\cos(\theta) = q_1 \cdot q_2 =$$
$$= s_1 s_2 + x_1 x_2 + y_1 y_2 + z_1 z_2$$

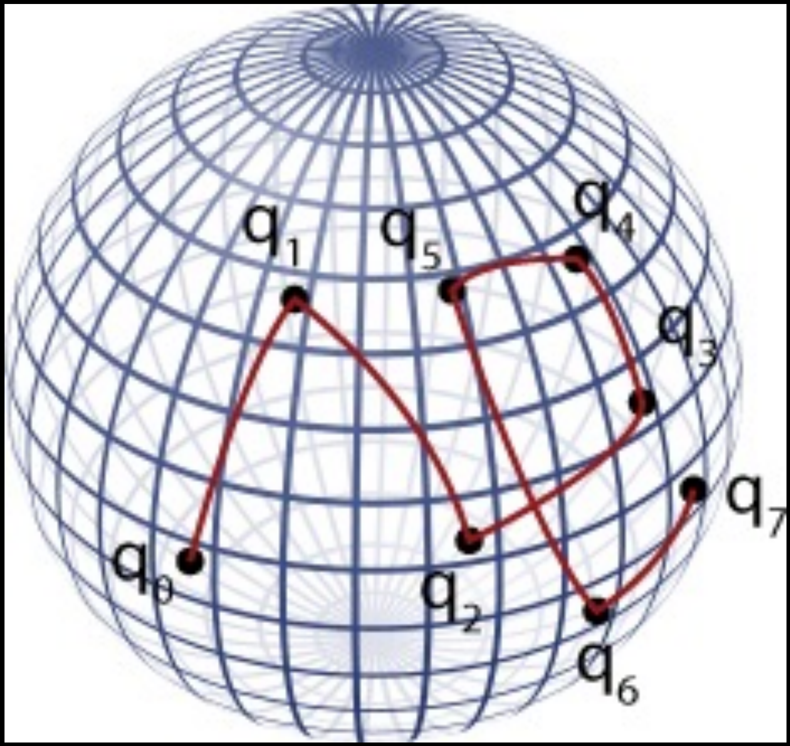
- u varies from 0 to 1
- $q_m = s_m + x_m \boldsymbol{i} + y_m \boldsymbol{j} + z_m \boldsymbol{k}$, for m = 1,2
- The above formula automatically produces a unit quaternion (not obvious, but true).

Interpolating more than two rotations

- Simplest approach:
  connect consecutive
  quaternions using SLERP

- Continuous rotations

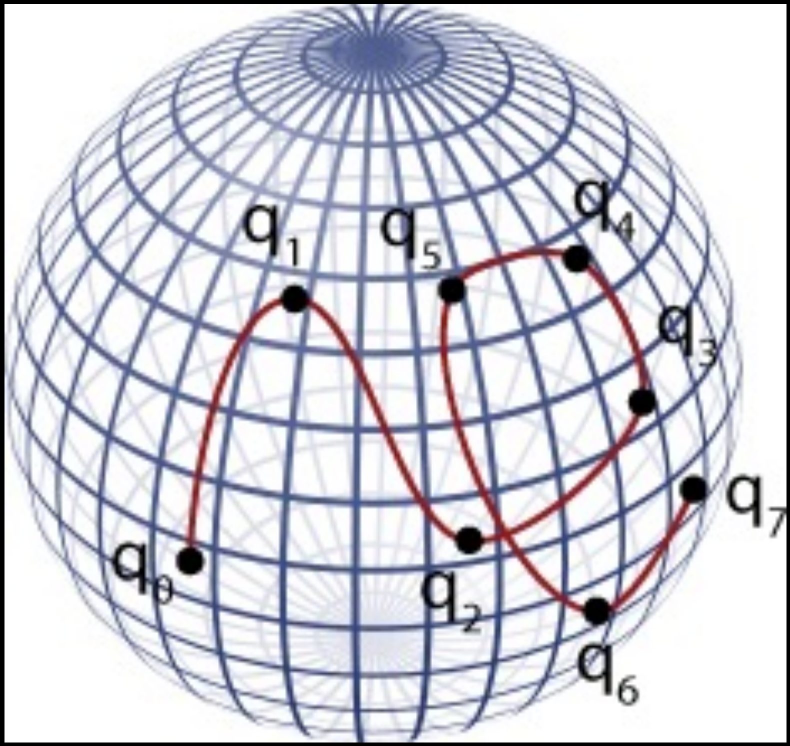
- Angular velocity
  not smooth at the joints



Interpolation with smooth velocities

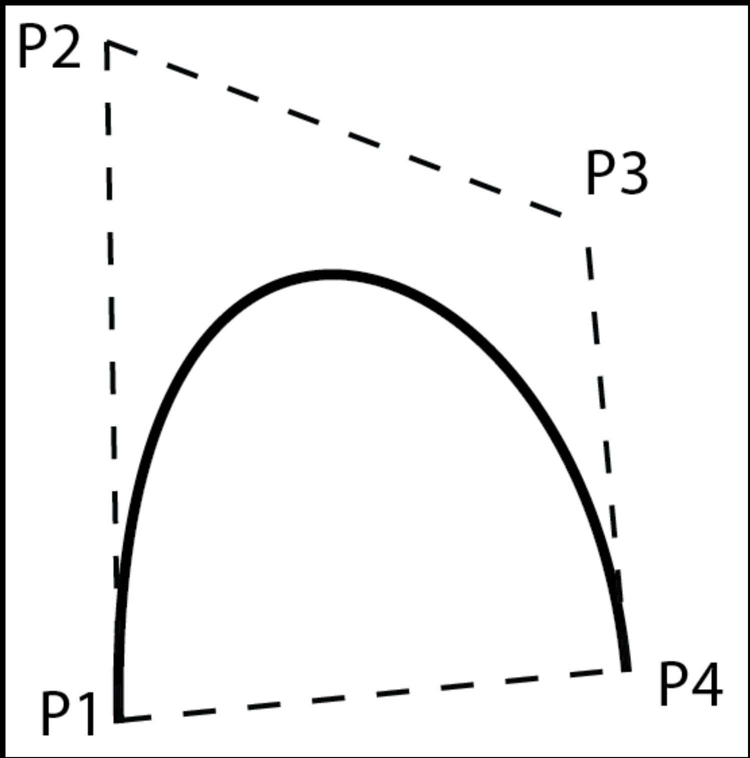
- Use splines on the unit quaternion sphere

- Reference: Ken Shoemake in the SIGGRAPH '85 proceedings (Computer Graphics, V. 19, No. 3, P. 245)



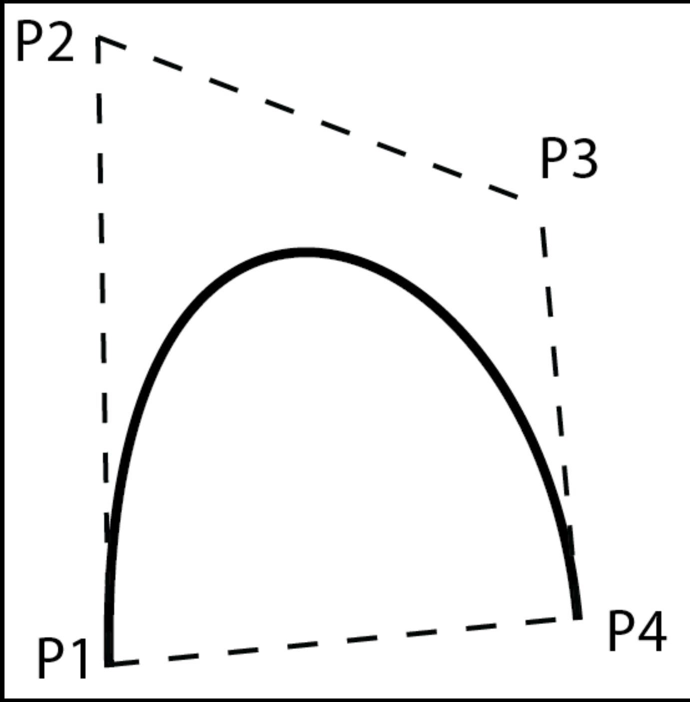
Bezier Spline

- Four control points
  - points P1 and P4 are on the curve
  - points P2 and P3 are off the curve;
    they give curve tangents at beginning and end



Bezier Spline

- $p(0) = P1$, $p(1) = P4$,
- $p'(0) = 3(P2-P1)$
- $p'(1) = 3(P4 - P3)$
- Convex Hull property: curve contained within the convex hull of control points
- Scale factor "3" is chosen to make "velocity" approximately constant

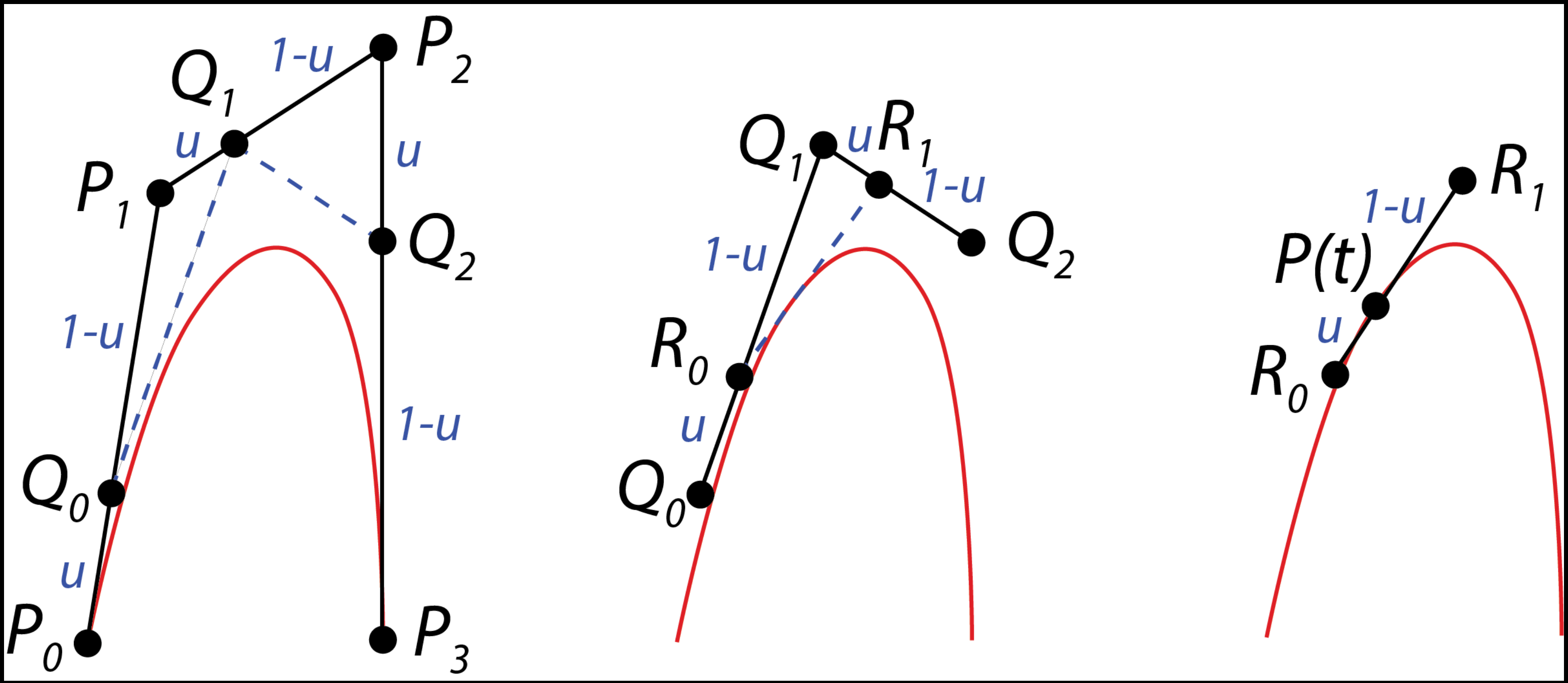


The Bezier Spline Formula

$$
\begin{bmatrix} x & y & z \end{bmatrix} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}
\underbrace{\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\text{Bezier basis}}
\underbrace{\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}}_{\text{Bezier control matrix}}
$$

- [x,y,z] is point on spline corresponding to u
- u varies from 0 to 1
- P1 = $[x_1\ y_1\ z_1]$ P2 = $[x_2\ y_2\ z_2]$
- P3 = $[x_3\ y_3\ z_3]$ P4 = $[x_4\ y_4\ z_4]$

DeCasteljau Construction

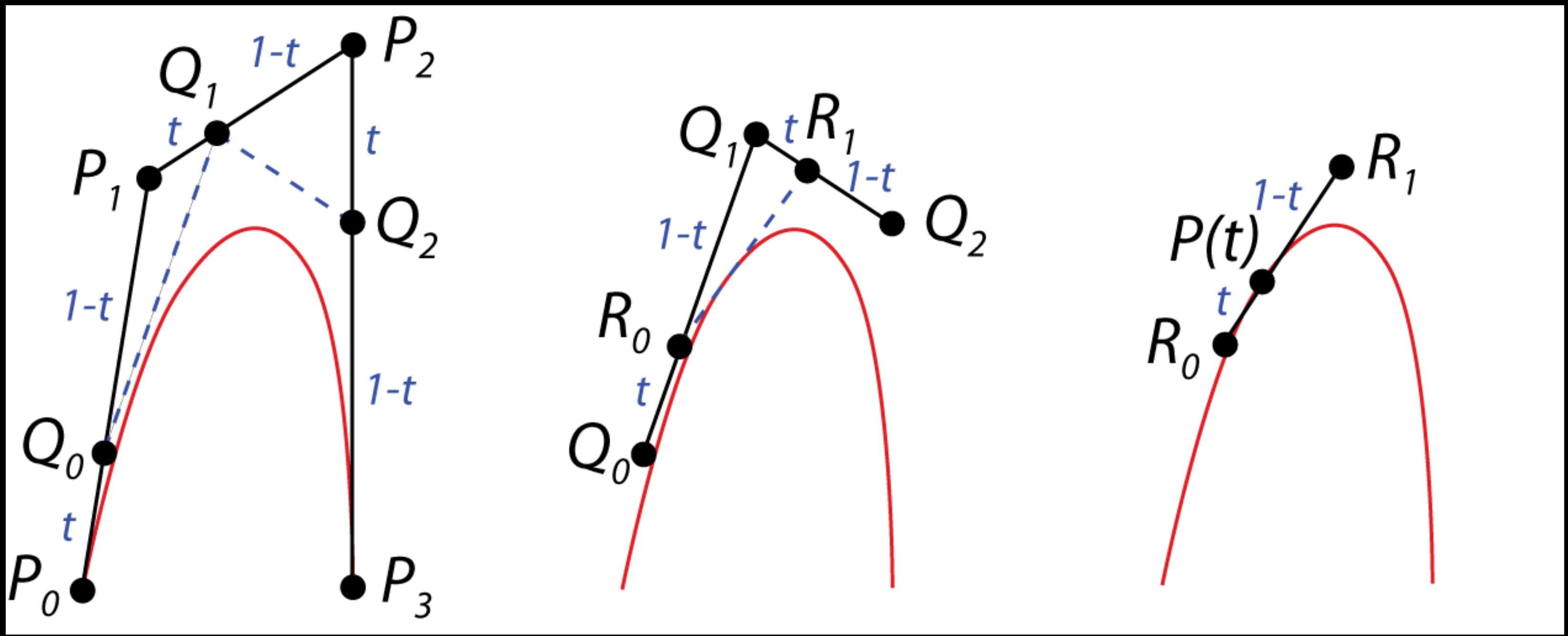


Efficient algorithm to evaluate Bezier splines.

Similar to Horner rule for polynomials.

Can be extended to interpolations of 3D rotations.

DeCasteljau on Quaternion Sphere



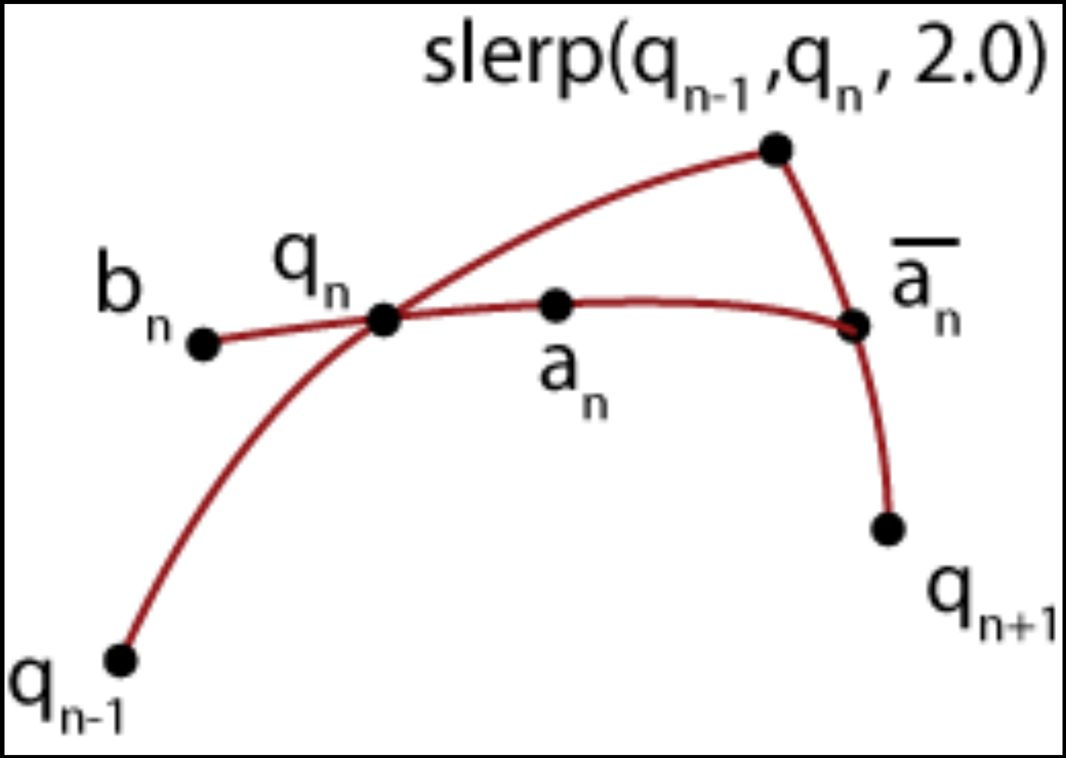
Given t, apply DeCasteljau construction:

Q_0 = Slerp(P_0,P_1,t) Q_1 = Slerp(P_1,P_2,t)
Q_2 = Slerp(P_2,P_3,t) R_0 = Slerp(Q_0,Q_1,t)
R_1 = Slerp(Q_1,Q_2,t) P(t)= Slerp(R_0,R_1,t)

Bezier Control Points for Quaternions

- Given quaternions q_{n-1}, q_n , q_{n+1} , form:

$\overline{a_n}$ = Slerp(Slerp(q_{n-1}, q_n, 2.0), q_{n+1}, 0.5)
a_n = Slerp(q_n , $\overline{a_n}$, 1.0 / 3)
b_n = Slerp(q_n , $\overline{a_n}$, -1.0 / 3)



Interpolating Many Rotations on Quaternion Sphere

- Given quaternions q_1, ..., q_N ,
  form Bezier spline control points (previous slide)

- Spline 1: q_1, a_1, b_2, q_2
- Spline 2: q_2, a_2, b_3, q_3 etc.

- Need a_1 and b_N; can set
  a_1 = Slerp(q_1, Slerp(q_3, q_2, 2.0), 1.0 / 3)
  b_N = Slerp(q_N, Slerp(q_{N-2}, q_{N-1}, 2.0), 1.0 / 3)

- To evaluate a spline at any t, use DeCasteljau construction