Orientation & Quaternions

Orientation

Position and Orientation

- The position of an object can be represented as a translation of the object from the origin
- The orientation of an object can be represented as a rotation of an object from its original unrotated orientation.

Position

- Cartesian coordinates (x,y,z) are an easy and natural means of representing a position in 3D space
- Of course, there are many other frames such as polar notation (r,θ,φ)

Orientation

- Several ways to represent a rotation:
  - Euler angles
  - Rotation vectors (axis/angle)
  - 3x3 matrices
  - Quaternions
  - …

Direct Matrix Representation

- Matrices are how we apply rotations to geometric data, so generally orientation representations need to be converted to matrix form to actually perform the rotation specified
- Why consider other representations?
  - Numerical issues
  - Storage issues
  - User interaction issues
  - Interpolation issues

Direct Matrix Representation

- Recall that an orthonormal matrix performs an arbitrary rotation.
- Given 3 mutually orthogonal unit vectors:

$$\mathbf{a} = \mathbf{b} \times \mathbf{c} \quad \mathbf{b} = \mathbf{c} \times \mathbf{a} \quad \mathbf{c} = \mathbf{a} \times \mathbf{b}$$

$$|\mathbf{a}| = |\mathbf{b}| = |\mathbf{c}| = 1$$

- A rotation of **a** onto the *x* axis, **b** onto the *y* axis, and **c** onto the *z* axis is performed by:

$$\begin{bmatrix} a_x & a_y & a_z & 0 \\ b_x & b_y & b_z & 0 \\ c_x & c_y & c_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Euler's Theorem

- Euler's Theorem: Any two independent orthonormal *n*-dimensional coordinate frames can be related by a sequence of no more than *n* rotations about basis vectors (coordinate axes) such that consecutive rotations are about distinct basis vectors.
- Leonhard Euler (1707-1783)
- Nothing to do with Euler integration, Newton-Euler dynamics, Euler's Formula, Euler equations, Euler characteristic…

Euler Angles

- We can represent an orientation in 3-d Euclidean space with 3 numbers
- Such a sequence of rotations around basis vectors is called an *Euler Angle Sequence*
- We'll normally use the sequence **ijk** (x y z)
- But we could also use:

ikj	**iji**	**iki**	**jik**
jki	**jij**	**jkj**	**kij**
kji	**kik**	**kjk**	

Matrix for Euler Angles

Matrix for our canonical **ijk** ordering:

$$\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos_x & \sin_x & 0 \\ 0 & -\sin_x & \cos_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos_y & 0 & -\sin_y & 0 \\ 0 & 1 & 0 & 0 \\ \sin_y & 0 & \cos_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos_z & \sin_z & 0 & 0 \\ -\sin_z & \cos_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

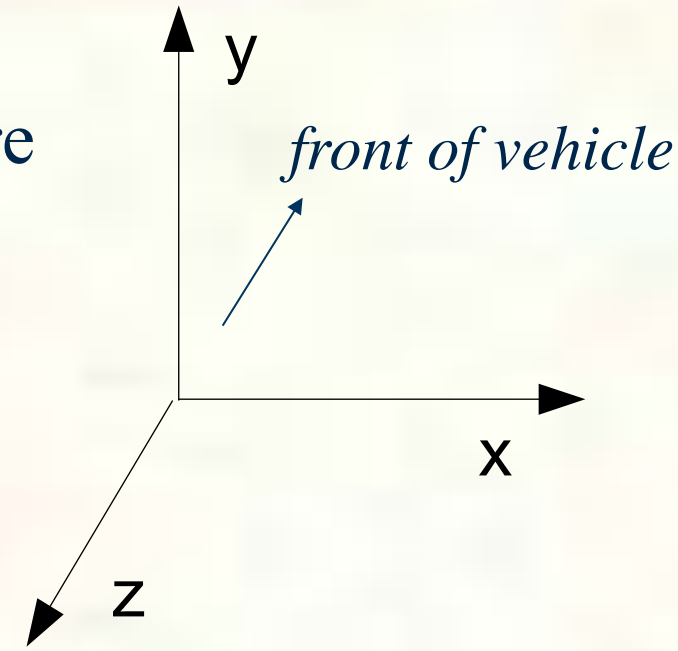
$$= \begin{bmatrix} c_y c_z & c_y s_z & -s_y & 0 \\ s_x s_y c_z - c_x s_z & s_x s_y s_z + c_x c_z & s_x c_y & 0 \\ c_x s_y c_z + s_x s_z & c_x s_y s_z - s_x c_z & c_x c_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Local vs. World Coordinates

- Matrix multiplication is not commutative - the order of operations is important
- Rotations are normally assumed to be relative to world coordinate axes, not local object coordinates
- Reversing the sequence order gives a local object coordinate representation

Vehicle Orientation

- Generally, for vehicles, it is most convenient to rotate in roll (z), pitch (y), and then yaw (x)
- This is a local coordinate representation
- Note that it's the reverse of our canonical world coordinate order
- This is quite intuitive where we have a well-defined up direction



Gimbal Lock

- A common problem with Euler angles is *gimbal lock*
- This results when two axes coincide after a rotation by some integer multiple of 90^{o} about a third axis, resulting in a singularity, i.e. a loss of a degree of freedom
- What is the longitude at the north or south pole?

Interpolating Euler Angles

- One can simply interpolate between the three values independently
- This will result in the interpolation following a different path depending on which of the 12 schemes you choose
- Interpolating near the ‘poles’ can be problematic

Pros and Cons of Euler Angles

- Pro
  - Compact (only 3 numbers)
- Con
  - Do not interpolate in a consistent way (pro or con)
  - Gimbal lock
  - Not simple to concatenate rotations

Axis/Angle Representation

- Euler's Theorem shows that any two orientations can be related by a single rotation about some axis vector (not necessarily a basis vector)
- This means that we can represent an arbitrary orientation as a rotation about some unit axis vector by some angle (4 numbers)

Axis/Angle Representation

- Storing an orientation as an axis and an angle uses 4 numbers, but Euler's Theorem says that we only need 3 numbers to represent an orientation
- Thus there is redundant information, the magnitude of the axis vector, in the axis/angle representation,
- Normalizing the axis vector constrains the extra degree of freedom since if we know the vector is unit length, we can get its third direction cosine if we know the other two.

Axis/Angle (OpenGL) Rotation Matrix

Given arbitrary unit axis vector $\mathbf{a}=(a_x,a_y,a_z)$ and counterclockwise rotation angle θ:

$$\begin{bmatrix} a_x^2+\mathrm{c}_\theta(1-a_x^2) & a_xa_y(1-\mathrm{c}_\theta)+a_z\mathrm{s}_\theta & a_xa_z(1-\mathrm{c}_\theta)-a_y\mathrm{s}_\theta & 0 \\ a_xa_y(1-\mathrm{c}_\theta)-a_z\mathrm{s}_\theta & a_y^2+\mathrm{c}_\theta(1-a_y^2) & a_ya_z(1-\mathrm{c}_\theta)+a_x\mathrm{s}_\theta & 0 \\ a_xa_z(1-\mathrm{c}_\theta)+a_y\mathrm{s}_\theta & a_ya_z(1-\mathrm{c}_\theta)-a_x\mathrm{s}_\theta & a_z^2+\mathrm{c}_\theta(1-a_z^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quaternions

Quaternions

- Quaternions are an extension of complex numbers that provide a way of rotating vectors just as vectors translate points
- Discovered by Hamilton in 1843 (and Gauss in 1819, but he didn't publish)
- For graphics, they are most useful as a means of representing orientations (rotations)

Quaternions

- Quaternions are an extension of complex numbers with 3 square roots of -1 (*ijk*) instead of just *i*
- The first component is a scalar real number, the other 3 form a vector in right-handed *ijk* space

$$\mathbf{q} = s + iq_1 + jq_2 + kq_3 \quad \text{where} \quad i^2 = j^2 = k^2 = ijk = -1$$

- or you can write it explicitly as a scalar and a vector

$$\mathbf{q} = \langle s, \mathbf{v} \rangle \quad \text{where} \quad \mathbf{v} = \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}$$

Unit Quaternions

- For representing rotations or orientations, 4 numbers is once again 1 too many, so as with axis/angle we use only unit length quaternions

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

- These correspond to the set of vectors that form the hypersurface of a 4D hypersphere of radius 1
- The hypersurface is actually a 3D volume in 4D space, but it can sometimes be visualized as an extension to the concept of a 2D surface on a 3D sphere

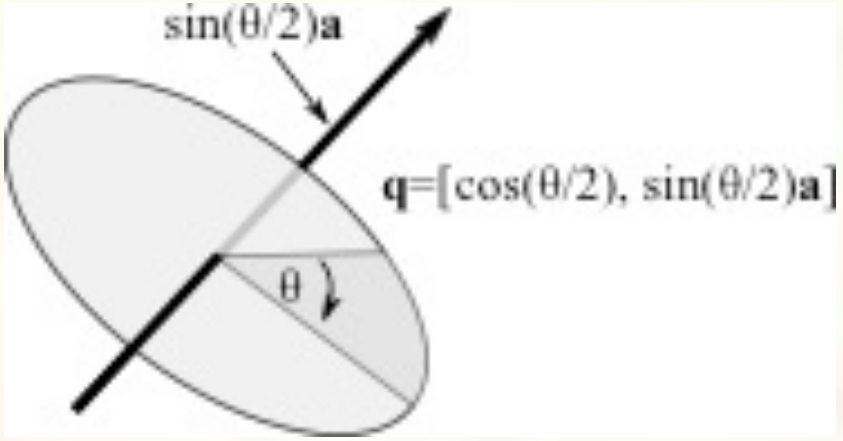
Unit Quaternions as Rotations

- A unit quaternion represents a rotation by an angle θ around a unit axis vector **a** as:

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

or

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



- If **a** is unit length, **q** is too

Unit Quaternions as Rotations

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

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

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

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

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

Conjugation Performs Rotation

- Quaternions can represent vectors by setting the scalar part to 0 (i.e. the axis vector with 0 rotation). $\mathbf{v} = \langle 0, \mathbf{v} \rangle$
- This vector (quaternion) needn't be unit length.
- Rotate the vector counterclockwise by angle θ about axis $\mathbf{a}$ by conjugating it with a unit quaternion representing the rotation

$$\mathbf{v}' = \mathbf{q}\mathbf{v}\mathbf{q}^{-1} \quad \mathbf{q} = \left\langle \cos\frac{\theta}{2}, \mathbf{a}\sin\frac{\theta}{2} \right\rangle \quad \mathbf{v}' = \langle 0, \mathbf{v}' \rangle$$

where

$$\mathbf{q}^{-1} = \frac{\left\langle \cos\frac{-\theta}{2}, \mathbf{a}\sin\frac{-\theta}{2} \right\rangle}{|\mathbf{q}|^2} = \left\langle \cos\frac{\theta}{2}, -\mathbf{a}\sin\frac{\theta}{2} \right\rangle$$

Quaternion Multiplication

- We can perform multiplication on quaternions if we expand them into their complex number form

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

- 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 (in particular it is not commutative)

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

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

Quaternion Multiplication

- Note that, just like complex numbers, two unit quaternions multiplied together will result in another unit quaternion
- 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
- So, in summary, multiplying unit quaternions in a particular order results in a unit quaternion that does the rotation that is performed by the two original rotations in that order.

Unit Quaternion to Matrix

- Matrix for unit quaternion:

$$\begin{bmatrix} 1-2q_2^2-2q_3^2 & 2q_1q_2+2sq_3 & 2q_1q_3-2sq_2 & 0 \\ 2q_1q_2-2sq_3 & 1-2q_1^2-2q_3^2 & 2q_2q_3+2sq_1 & 0 \\ 2q_1q_3+2sq_2 & 2q_2q_3-2sq_1 & 1-2q_1^2-2q_2^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Quaternion Interpolation

Linear Interpolation

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

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

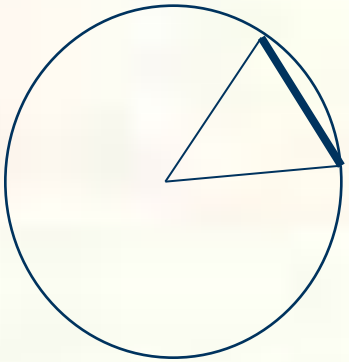
where t ranges from 0 to 1

- This is a (convex) affine combination of points
- It can of course also be written

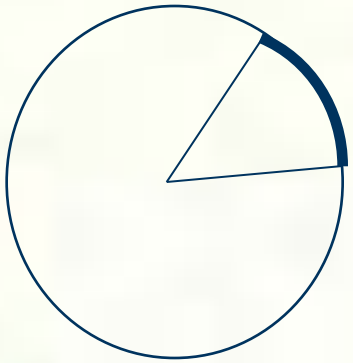
$$\text{Lerp}(t,\mathbf{a},\mathbf{b}) = \mathbf{a} + t(\mathbf{b}-\mathbf{a})$$

Spherical Linear Interpolation

- If we want to interpolate between two points on a sphere (or hypersphere), we 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

- Geometrically, the spherical linear interpolation of two unit vectors in N dimensional space is given by:

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

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

Quaternion Interpolation

- Remember that there are two redundant vectors in quaternion space for every unique orientation in 3D space
- What is the difference between:

  Slerp(t,**a**,**b**) and Slerp(t,-**a**,**b**) ?

- One of these will travel less than 90 degrees while the other will travel more than 90 degrees across the sphere
- This corresponds to rotating the ‘short way’ or the ‘long way’
- Usually, we want to take the short way, so we negate one of them if their dot product is < 0

Quaternion Summary

- Quaternions are 4D vectors that can represent 3D rigid body orientations
- We use unit quaternions for orientations (rotations)
- Quaternions are more compact than matrices to represent rotations/orientations
- Key operations:
  - Quaternion multiplication: faster than matrix multiplication for combining rotations
  - Quaternion conjugation: faster than matrix vector multiplication for performing rotations
  - Quaternion to matrix: to combine quaternion rotations with other affine transforms
  - Slerp: to interpolate between arbitrary orientations