Rotations and Orientation

Position and Orientation

The position of an object can be represented as a translation of the object from the origin

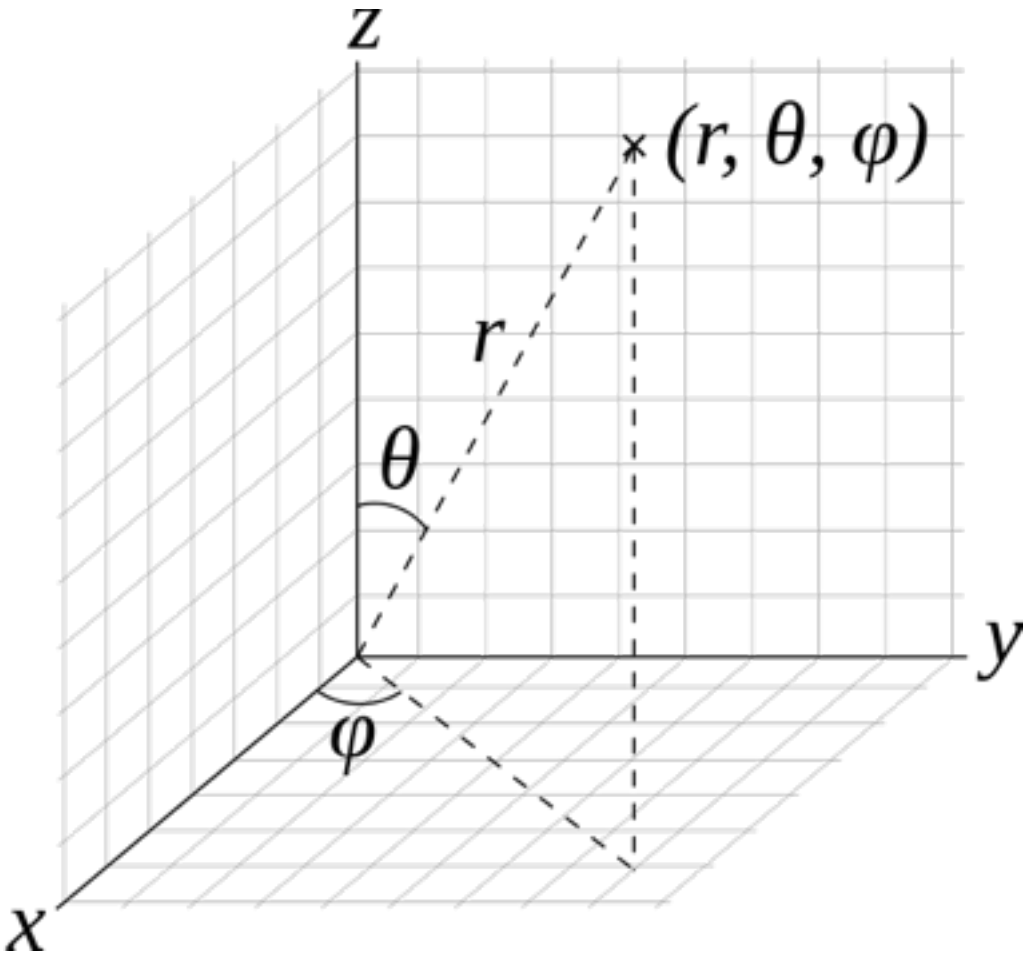
The orientation of an object can be represented as a rotation of an object from its original unrotated orientation.

Position

Cartesian coordinates (x,y,z) are an easy and natural means of representing a position in 3D space

…But there are many other representations such as spherical coordinates (r,θ,ϕ)

Spherical Coordinates



Orientation

Many ways to represent a rotation:

- 3x3 matrices
- Euler angles
- Rotation vectors (axis/angle)
- Quaternions

Why might multiple representations be useful?

Uses for Other Representations

Numerical issues

Storage

User interaction

Interpolation

Euler's Rotation Theorem

"An arbitrary rotation may be described by only three parameters" (Wolfram definition)

i.e. the composition of multiple rotations is a rotation

Euler Angles

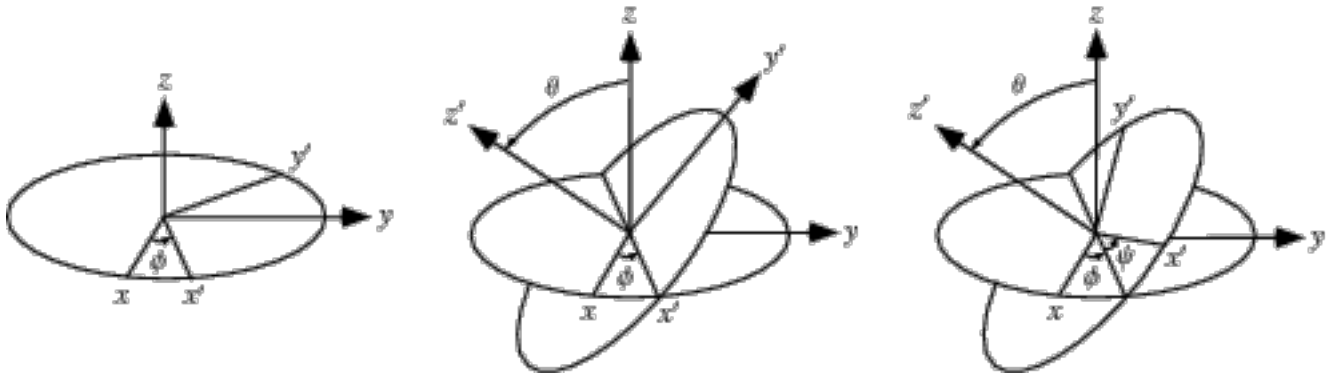
- We can represent an orientation in 3D Euclidean space with three numbers
- This sequence of rotations around basis vectors is called an *Euler Angle Sequence*

Euler Angle Sequences

Often a sequence like (x y z) is used:

- Rotate about x axis, then y axis, then z axis

But any sequence works!



(Rotation about z, x' and z')

Note: Tait-Bryan vs Proper Euler

Tait-Bryan rotations rotate about three distinct axes (x y z)

Proper Euler angles share axis for first and last rotation (z x z)

- Both systems can represent all 3D rotations
- Tait-Bryan common in engineering applications, so we'll use those…

Note: Intrinsic vs Extrinsic Rotations

Intrinsic rotations apply to axis in rotated coordinate system

- Coordinate system of next rotation relative to previous rotation

Extrinsic rotations apply to axis in world coordinate system

- Coordinate system of next rotation relative to (fixed) world coordinate system

We'll Use Extrinsic Rotations!

Rotations generally assumed to be extrinsic in computer graphics, but consider rotation (z y x)…

How do we apply this rotation intrinsically?

Intrinsic Euler Angles

1) Rotation about x axis:

$x' = x$

$y' = y\cos(\alpha) - z\sin(\alpha)$

$z' = y\sin(\alpha) + z\cos(\alpha)$

2) Rotation about y axis:

$x'' = x'\cos(\beta) + z'\sin(\beta)$

$y'' = y'$

$z'' = -x'\sin(\beta) + y'\cos(\beta)$

3) Rotation about z axis:

$x''' = x''\cos(\gamma) - y''\sin(\gamma)$

$y''' = x''\sin(\gamma) + y''\cos(\gamma)$

$z''' = z''$

Converting Between Intrinsic and Extrinsic

Turns out intrinsic rotation order is the reverse of extrinsic rotation order!

e.g. intrinsic rotation (x y z) is equivalent to extrinsic rotation (z y x)

extrinsic rotation (x y z) is equivalent to intrinsic rotation (z y x)

Matrix Representation

How we apply rotations to geometric data

Orientation representations often converted to matrix form to perform rotation

Rotations in Two Dimensions

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

x’ = xcos(Θ) - ysin(Θ)

y’ = xsin(Θ) + ycos(Θ)

Rotations in Three Dimensions

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

x' = x

y' = ycos(Θ) - zsin(Θ)

z' = ysin(Θ) + zcos(Θ)

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

x' = xcos(Θ) - zsin(Θ)

y' = y

z' = -xsin(Θ) + zcos(Θ)

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

x' = xcos(Θ) - ysin(Θ)

y' = xsin(Θ) + cos(Θ)

z' = z

A Matrix for Euler Angles

Matrix for (x y z) sequence:

$$R_x R_y R_z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos_x & -sin_x & 0 \\ 0 & sin_x & cos_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos_y & 0 & sin_y & 0 \\ 0 & 1 & 0 & 0 \\ -sin_y & 0 & cos_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos_z & -sin_z & 0 & 0 \\ sin_z & cos_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

General Matrix Representation

Orthonormal matrices perform arbitrary rotations

Given 3 mutually orthogonal unit vectors:

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

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

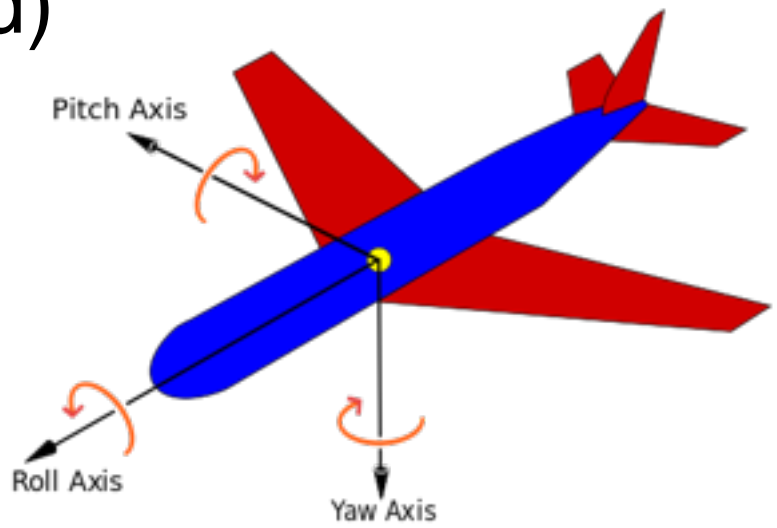
We can apply rotation of **a** onto the *x* axis, **b** onto the *y* axis and **c** onto the *z* axis using:

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

Yaw, Pitch, and Roll

Naming convention for rotations based on vehicle orientation

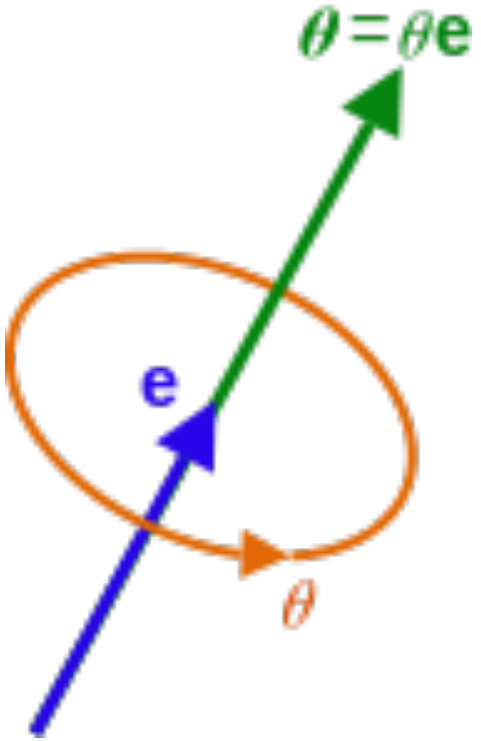
- Yaw along Z axis (below)
- Pitch along Y axis (right)
- Roll along X axis (forward)



Axis/Angle Representation

Parameterizes Euler's Theorem as a unit vector e = (e_x, e_y, e_z) and counterclockwise rotation angle Θ

Provides rotation direction and magnitude



Gimbal Lock

- Issue with Euler angles
- Occurs when two axes coincide after rotation by some integer multiple of 90° about a third axis
- Loss of a degree of freedom
- Consider: what is the longitude at the north or south pole?

Using a Fourth Gimbal

Providing a 4th gimbal that maintains large angle between roll and yaw can prevent gimbal lock

Gimbal lock an issue on Apollo 11 and 13 missions but 4th gimbal not used for weight reasons

https://www.youtube.com/watch?v=7vpRPbPs8j4

Gimbal Lock Video

https://www.youtube.com/watch?v=zc8b2Jo7mno

How can we solve this problem?

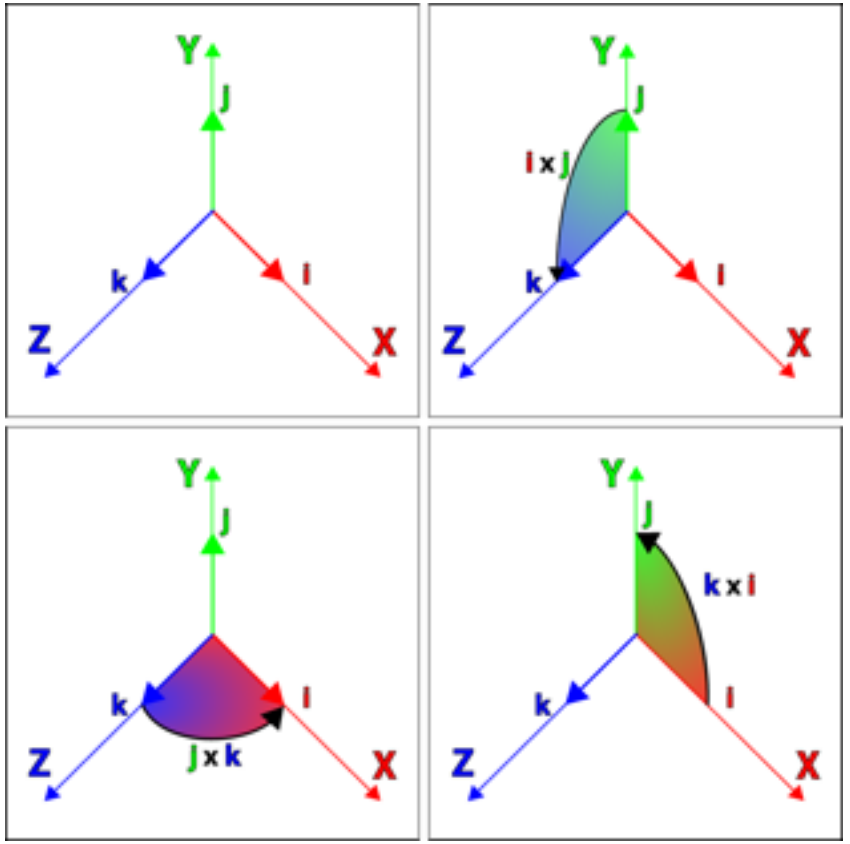
Quaternions

- Extension of complex numbers that provide a way of rotating vectors
- Discovered by Hamilton in 1843 (and Gauss in 1819 but he didn't publish)
- Often used in graphics for representing orientation and rotation

How Do They Work?

Quaternions are an extension of complex numbers with 3 square roots of -1

- (i j k) instead of just i



Quaternion Representation

- First component is a scalar real number
- Other three components form a vector in right-handed *ijk* space:

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

where

$$i^2 = j^2 = k^2 = ijk = -1$$

Unit Quaternions

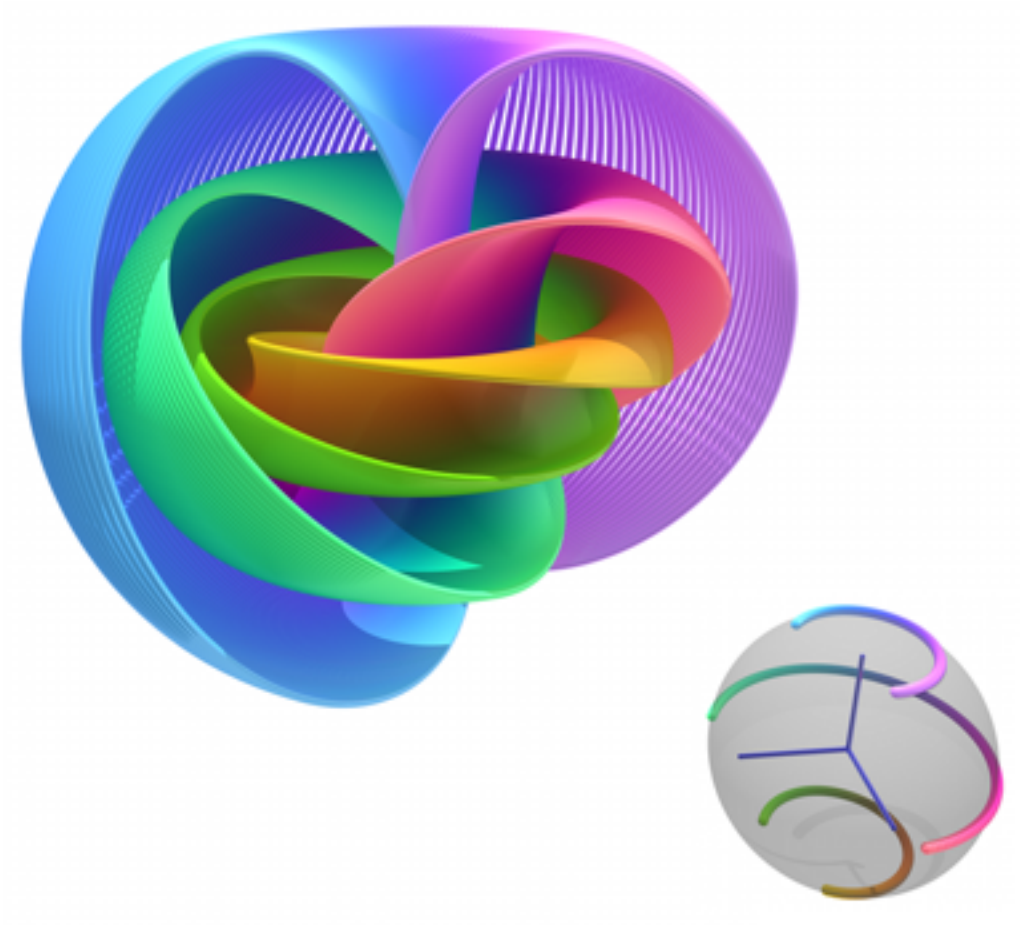
As in axis/angle representation, can use unit length quaternion for orientation:

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

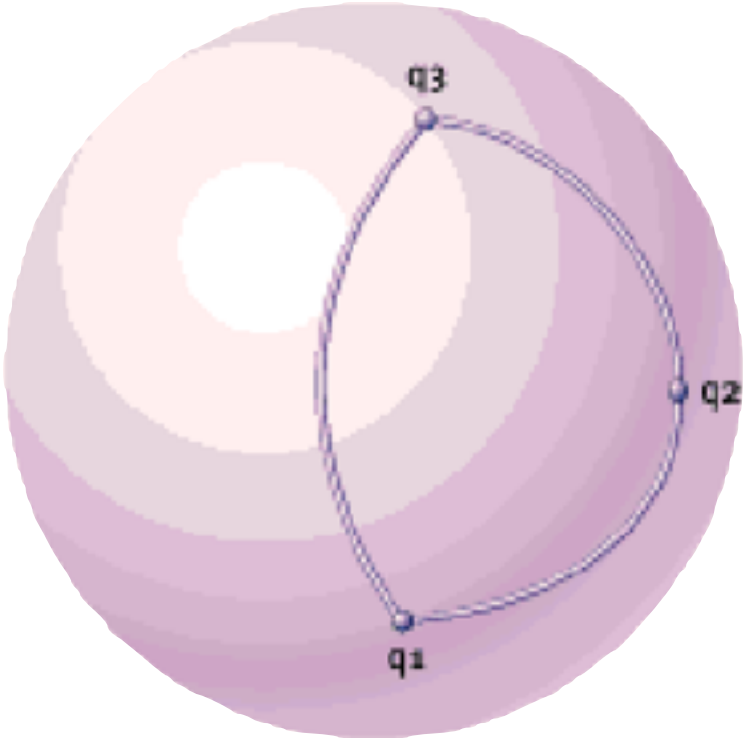
Represents a set of vectors forming a hypersurface of 4D hypersphere of radius 1

Hypersurface is a 3D volume in 4D space, but think of it as the same idea of a 2D surface on a 3D sphere

A Quaternion Visualization…



A More Understandable Representation…



Other Notations

This can also be written explicitly as a scalar-vector pair:

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

Or a rotation by an angle about an axis:

$$q = [cos\tfrac{1}{2}\theta, sin\tfrac{1}{2}\theta\hat{v}]$$

Quaternion Multiplication

- Unit quaternions multiplied together create another unit quaternion
- Multiplication by a complex number is a rotation in the complex plane
- Quaternions extend planar rotations of complex numbers to 3D rotations in space

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

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

Converting Euler Angles to Quaternions

$q = q_{yaw} q_{pitch} q_{roll}$ where:

$q_{yaw} = \langle \cos(\gamma/2), [0, 0, \sin(\gamma/2)] \rangle$

$q_{pitch} = \langle \cos(\beta/2), [0, \sin(\beta/2), 0] \rangle$

$q_{roll} = \langle \cos(\alpha/2), [\sin(\alpha/2), 0, 0] \rangle$

Note that quaternion multiplication is not commutative

Converting Quaternions to Euler Angles

$$\alpha = \operatorname{atan2}(2(sq_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2))$$

$$\beta = \operatorname{asin}(2(sq_2 - q_1q_3))$$

$$\gamma = \operatorname{atan2}(2(sq_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2))$$

Remember Linear Interpolation?

`lerp(t, a, b) = (1-t)a + (t)b`

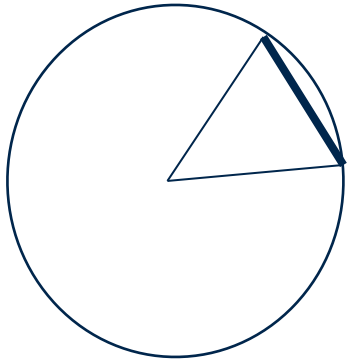
linearly interpolates between points **a** and **b** where 0 <= t <= 1

Also possible to write as

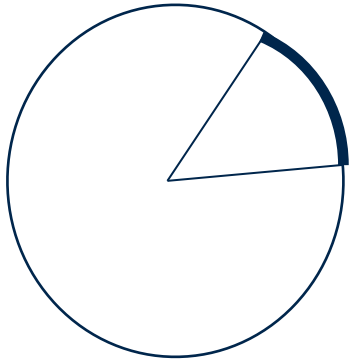
`lerp(t, a, b) = a + t(b-a)`

Spherical Linear Interpolation

Lerps won't work on a sphere (or hypersphere):



Must travel along surface of sphere following the great arc:



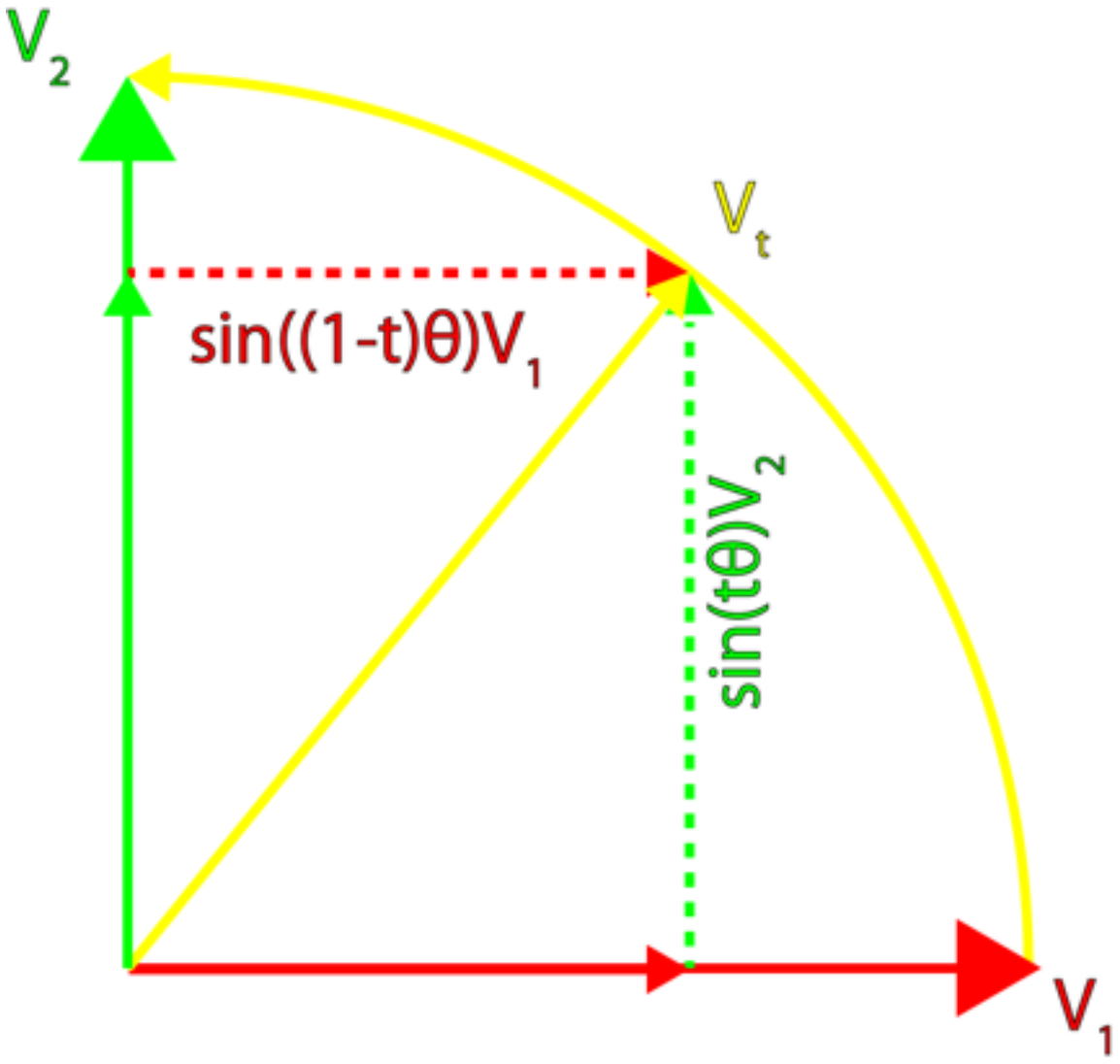
SLERP

$$q_t = \frac{sin((1-t)\theta)}{sin(\theta)} q_1 + \frac{sin(t\theta)}{sin(\theta)} q_2$$

where q1 and q2 are orientations for points a and b along parameter t and $\Theta = \cos^{-1}(a \cdot b)$

Note that when the angular distance between points is small, sin(Θ) approaches 0.

Must switch back to LERP


V2
Vt
sin((1-t)θ)V1
sin(tθ)V2
V1

Quaternion Interpolation

Two redundant vectors in quaternion space for every unique orientation in 3D space:

- `slerp(t, a, b)` and `slerp(t, -a, b)` end up at the same place
- …but one travels < 90° and one travels > 90°

To take the short way, negate one orientation if quaternion dot product < 0

SQAD

Spherical and Quadrangle

Smoothly interpolates over a path of rotations

Defines “helper” quaternion to act as a control point

Using Quaternions

OpenGL can't work directly with quaternions

Also they're difficult to specify in terms of rotations

General practice is to convert Euler angles to quaternions for interpolation only

- Most (if not all) game/graphics engines are doing this under the hood!

Quaternion Summary

- 4D vectors that represent 3D rigid body orientations
- More compact than matrices for representing rotations/orientations
- Free from Gimbal lock
- Can convert between quaternion and matrix representation
- SLERP allows interpolation between arbitrary orientations

Additional Reading

https://www.3dgep.com/understanding-quaternions/

http://www.gamasutra.com/view/feature/131686/rotating_objects_using_quaternions.php