GAME2016
Mathematical Foundation of Game Design and Animation

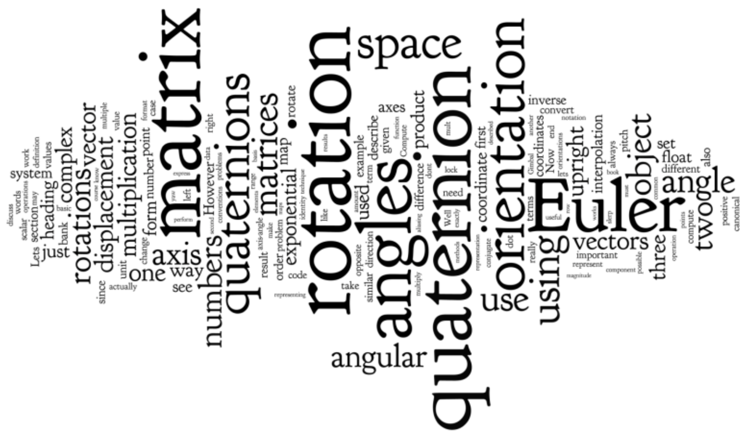
Lecture 7

Rotation in Three Dimensions

Dr. Paolo Mengoni
pmengoni@hkbu.edu.hk
Senior Lecturer @HKBU Department of Interactive Media

Agenda

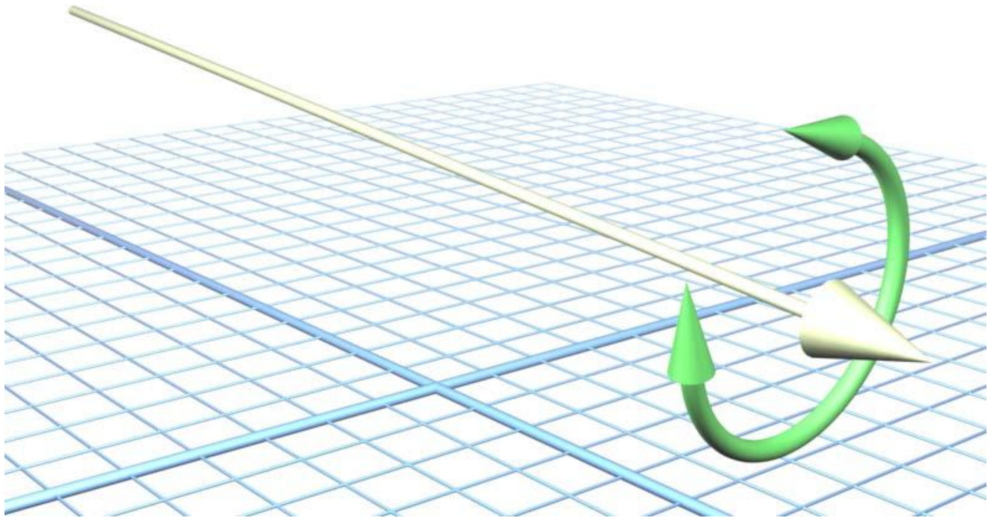
- Orientation, direction, and angular displacement.
- How to express orientation:
  - Matrix form
  - Euler angles
  - The axis-angle and exponential map forms
  - Quaternions at a glance
- Comparison between the different methods to express orientation.
  - How to choose representation
  - Quick note on conversion



What Exactly is “Orientation”?

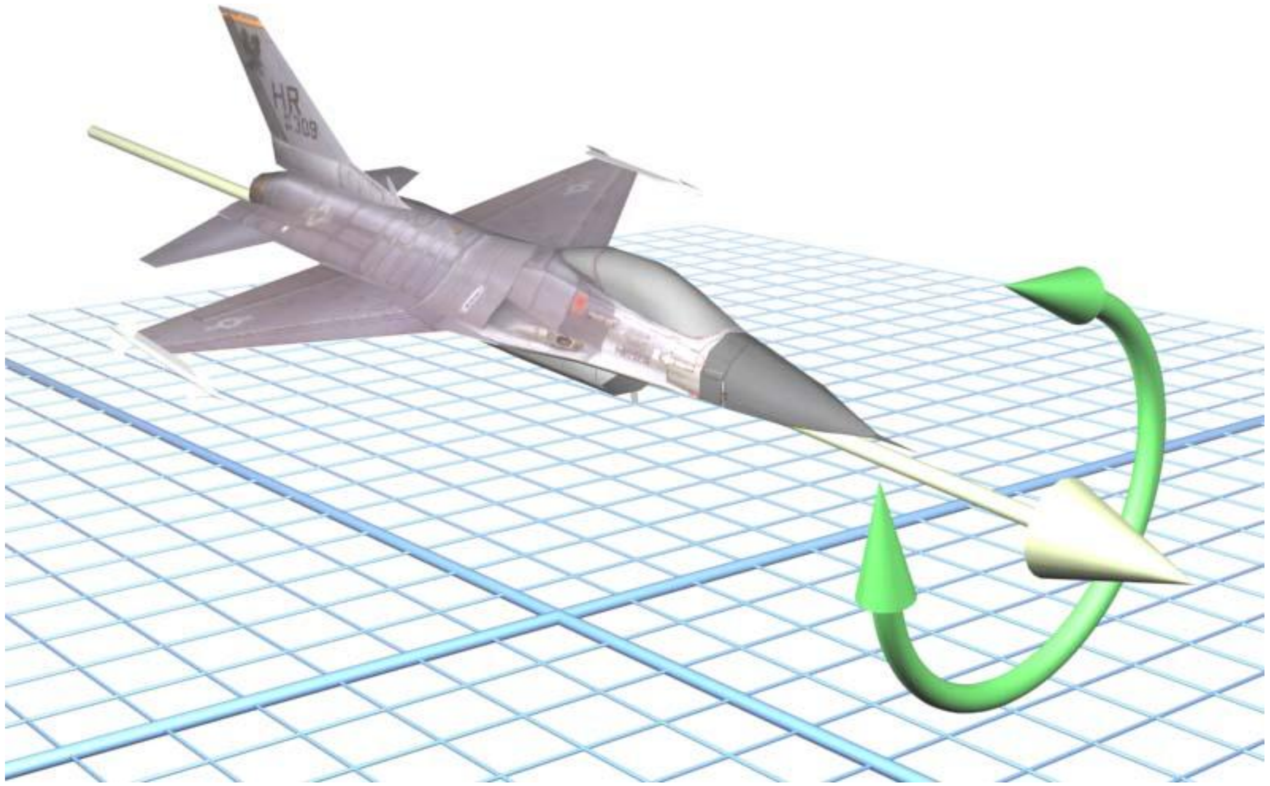
Orientation

- What is orientation? More than direction.
- A vector specifies direction, but it can also be twisted.



This is Important Because

- Twisting an object changes its orientation.



Direction and Orientation

- Specifying a direction using two angles.
  - Using polar coordinates.
- Specifying an orientation requires 3 angles.
  - Or at least 3 numbers whichever way you represent it.
- There are 3 popular ways to represent orientation.
  - Matrices
  - Euler angles
    - Axis Angles and Exponential Map
  - Quaternions

Angular Displacement

- Orientation can’t be given in absolute terms.
- Orientation is a rotation from some known reference orientation (often called the *identity* or *home* orientation).
  - Just as a position is a translation from some known point (i.e., from the origin)
- The amount of rotation is known as an *angular displacement*.

Matrix Form

Matrix Form

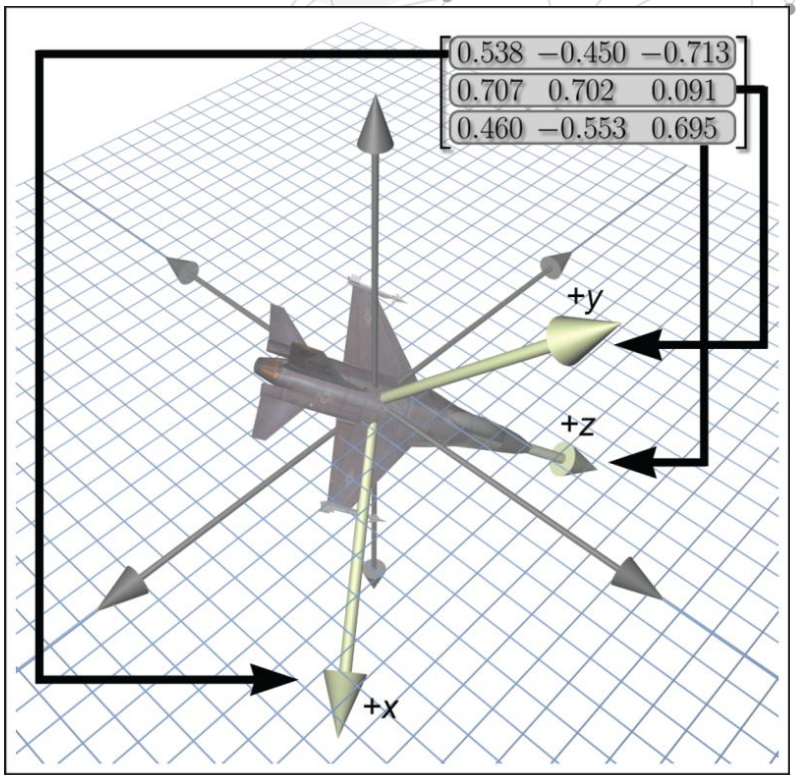
- List the relative orientation of two coordinate spaces by listing the transformation matrix that takes one space to another.
- For example: from object space to upright space.
- Transform back by using the inverse matrix.

Example

- We've seen how a matrix can be used to transform points from one coordinate space to another.

- In the figure, the matrix in the upper right hand corner can be used to rotate points from the object space of the jet into upright space.

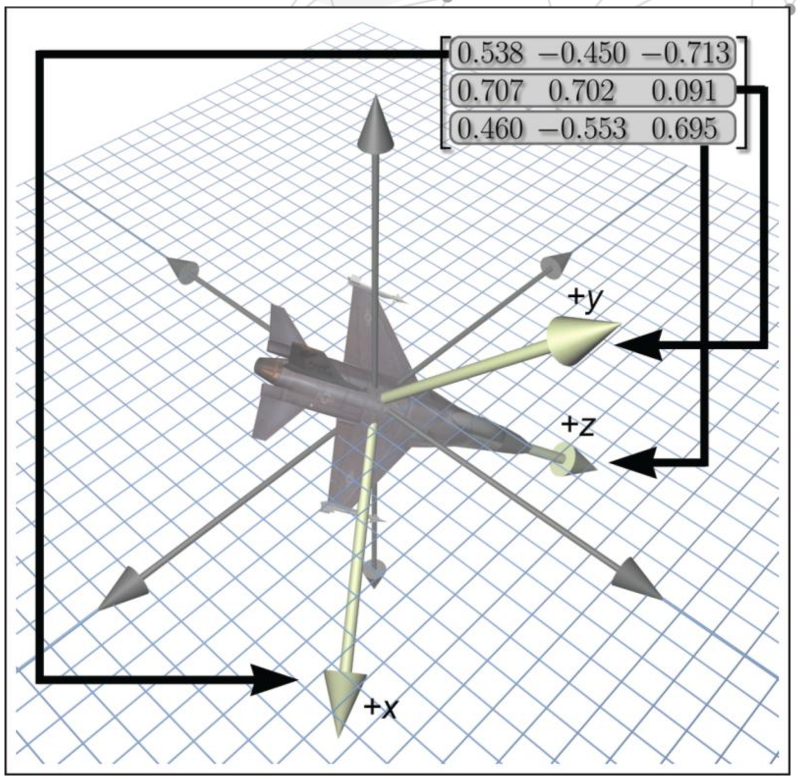
- Each row is in direct relationship to the coordinates for the jet's body axes.



$\begin{bmatrix} 0.538 & -0.450 & -0.713 \\ 0.707 & 0.702 & 0.091 \\ 0.460 & -0.553 & 0.695 \end{bmatrix}$
+y
+z
+x

Example

- The rotation matrix contains the object axes expressed in upright space.
- Simultaneously, it is a rotation matrix.
- We can multiply row vectors by this matrix to transform those vectors from object space coordinates to upright space coordinates.



$\begin{bmatrix} 0.538 & -0.450 & -0.713 \\ 0.707 & 0.702 & 0.091 \\ 0.460 & -0.553 & 0.695 \end{bmatrix}$
+y
+z
+x

Direction Cosines Matrix

- A *direction cosines matrix* is the same thing as a rotation matrix, but the term refers to a special way to interpret (or construct) the matrix.
- The term *direction cosines* refers to the fact that each element in a rotation matrix is the dot product of a cardinal axis in one space with a cardinal axis in the other space.
- For example, the center element m_{22} in a 3 x 3 matrix gives the dot product that the *y*-axis in one space makes with the *y*-axis in the other space.

Direction Cosines Matrix

- The basis vectors of a coordinate space are the mutually orthogonal unit vectors

$$\hat{\mathbf{p}},\ \hat{\mathbf{q}},\ \hat{\mathbf{r}},$$

- while a second coordinate space with the same origin has as its basis a different (but also orthonormal) basis

$$\hat{\mathbf{p}}',\ \hat{\mathbf{q}}',\ \hat{\mathbf{r}}'.$$

- The rotation matrix that rotates row vectors from the first space to the second is the matrix of direction cosines.

Direction Cosines Matrix

- The matrix of direction cosines (dot products) of each pair of basis vectors, is constructed as follows:

$$\mathbf{v}\begin{bmatrix} \hat{\mathbf{p}}\cdot\hat{\mathbf{p}}' & \hat{\mathbf{q}}\cdot\hat{\mathbf{p}}' & \hat{\mathbf{r}}\cdot\hat{\mathbf{p}}' \\ \hat{\mathbf{p}}\cdot\hat{\mathbf{q}}' & \hat{\mathbf{q}}\cdot\hat{\mathbf{q}}' & \hat{\mathbf{r}}\cdot\hat{\mathbf{q}}' \\ \hat{\mathbf{p}}\cdot\hat{\mathbf{r}}' & \hat{\mathbf{q}}\cdot\hat{\mathbf{r}}' & \hat{\mathbf{r}}\cdot\hat{\mathbf{r}}' \end{bmatrix} = \mathbf{v}'.$$

- These axes can be interpreted as geometric entities
  - Independent from coordinates used to describe the axes, the rotation matrix will be the same.

Example

- For example, let's say that our axes are described using the first coordinate space.

- $\hat{\mathbf{p}}, \hat{\mathbf{q}}, \hat{\mathbf{r}},$ have the trivial forms [1, 0, 0], [0, 1, 0] and [0, 0, 1], respectively.

- The basis vectors of the second space, $\hat{\mathbf{p}}', \hat{\mathbf{q}}', \hat{\mathbf{r}}'$ arbitrary coordinates.
  - They are not expressed in their own space.

Example

- When we substitute the trivial vectors $\hat{\mathbf{p}}, \hat{\mathbf{q}}, \hat{\mathbf{r}},$ into the matrix on the previous slide and expand the dot products, we get:

$$\begin{bmatrix} [1,0,0]\cdot\hat{\mathbf{p}}' & [0,1,0]\cdot\hat{\mathbf{p}}' & [0,0,1]\cdot\hat{\mathbf{p}}' \\ [1,0,0]\cdot\hat{\mathbf{q}}' & [0,1,0]\cdot\hat{\mathbf{q}}' & [0,0,1]\cdot\hat{\mathbf{q}}' \\ [1,0,0]\cdot\hat{\mathbf{r}}' & [0,1,0]\cdot\hat{\mathbf{r}}' & [0,0,1]\cdot\hat{\mathbf{r}}' \end{bmatrix}$$

$$= \begin{bmatrix} p'_x & p'_y & p'_z \\ q'_x & q'_y & q'_z \\ r'_x & r'_y & r'_z \end{bmatrix}$$

$$= \begin{bmatrix} -\hat{\mathbf{p}}'- \\ -\hat{\mathbf{q}}'- \\ -\hat{\mathbf{r}}'- \end{bmatrix}$$

Conclusion

- In other words, the rows of the rotation matrix are the basis vectors of the output coordinate space.
  - Expressed using the coordinates of the input coordinate space.
- This fact is true for rotation matrices, and for all transformation matrices.
  - This is the central idea of why a transformation matrix works.

The Other Case

- Instead of expressing all the basis vectors using the first coordinate space, measure them using the second coordinate space (the output space).

- This time, $\hat{\mathbf{p}}', \hat{\mathbf{q}}', \hat{\mathbf{r}}'$ have trivial forms, and $\hat{\mathbf{p}}, \hat{\mathbf{q}}, \hat{\mathbf{r}},$ are arbitrary.

The Other Case

- Putting these into the direction cosines matrix produces:

$$\begin{bmatrix} \hat{\mathbf{p}} \cdot [1,0,0] & \hat{\mathbf{q}} \cdot [1,0,0] & \hat{\mathbf{r}} \cdot [1,0,0] \\ \hat{\mathbf{p}} \cdot [0,1,0] & \hat{\mathbf{q}} \cdot [0,1,0] & \hat{\mathbf{r}} \cdot [0,1,0] \\ \hat{\mathbf{p}} \cdot [0,0,1] & \hat{\mathbf{q}} \cdot [0,0,1] & \hat{\mathbf{r}} \cdot [0,0,1] \end{bmatrix}$$

$$= \begin{bmatrix} p_x & q_x & r_x \\ p_y & q_y & r_y \\ p_z & q_z & r_z \end{bmatrix}$$

$$= \begin{bmatrix} | & | & | \\ \hat{\mathbf{p}}^T & \hat{\mathbf{q}}^T & \hat{\mathbf{r}}^T \\ | & | & | \end{bmatrix}$$

Conclusion

- This says that the columns of the rotation matrix are formed from the basis vectors of the input space, expressed using the coordinates of the output space.

- This is *not* true of transformation matrices in general; it applies only to orthogonal matrices such as rotation matrices.

Advantages of Matrix Form 1

Rotation of vectors is immediately available.

- You can use a matrix to rotate vectors between object and upright space.

- No other representation of orientation allows this to rotate vectors, you must convert the orientation to matrix form.

Advantages of Matrix Form 2

Format used by graphics APIs.

- Graphics APIs use matrices to express orientation.
  - When you are communicating with the graphics hardware using APIs you need to express your transformations as matrices eventually.
  - Need conversion between you program internal representation to matrices at some point in the graphics pipeline.

Advantages of Matrix Form 3

Concatenation of multiple angular displacements

- It is possible to collapse nested coordinate space relationships.
- For example, if we know the orientation of object A relative to object B, and we know the orientation of object B relative to object C, then using matrices, we can determine the orientation of object A relative to object C.
- Recall the nested coordinate spaces and how matrices can be concatenated.

Advantages of Matrix Form 4

Matrix inversion.

- When an angular displacement is represented in matrix form, it is possible to compute the opposite angular displacement using matrix inversion.
  - Rotation matrices are orthogonal.
  - The computation can be done by transposing the matrix.

Disadvantages of Matrix Form 1

Matrices take more memory.

- If we need to store many orientations (e.g., keyframes in an animation sequence), need extra space for nine numbers.
  - instead of three with other representations.
- Example. We are animating a model of a human that is broken up into 15 pieces for different body parts.
  - Animation is accomplished strictly by controlling the orientation of each part relative to its parent part.
- We are storing one orientation for each part, per frame, and our animation data is stored at a reasonable rate, say, 15fps.
  - Thus, we will have 225 orientations per second.
- With matrices and 32-bit FP numbers, each frame will take 8.1KB.
- With Euler angles (next topic), the same data would only take 2.7KB.
  - For a mere 30 seconds of animation data, matrices would take 162K more than the same data stored using Euler angles!

Disadvantages of Matrix Form 2

Difficult for humans to use.

- Matrices are not intuitive for humans to work with directly.
  - Too many numbers, and they are all from –1 to 1.

- Humans naturally think about orientation in terms of angles, but a matrix is expressed using vectors.

Disadvantages of Matrix Form 3

Matrices can be malformed.

- A matrix uses nine numbers, when only three are necessary.
  - A matrix contains six degrees of redundancy.
- There are six constraints that must be satisfied for a matrix to be valid to represent an orientation.
- The rows must be unit vectors, and they must be mutually perpendicular.

How Can Matrices Get Malformed?

- Matrix that contains scale, skew, reflection, or projection.
  - Any non-orthogonal matrix is not a well-defined rotation matrix.
  - Reflection matrices (which are orthogonal) are not valid rotation matrices either.
- Bad data from an external source.
  - For example, motion capture can produce errors due to the capturing process.
  - Many modeling packages are notorious for producing malformed matrices.
- We may create bad data due to floating point rounding errors.
  - For example, applying a large number of incremental changes to an orientation leads to large number of matrix multiplications which leads to errors due to the limited floating point precision.

Summary of Matrix Form 1

- Matrices are a brute force method of expressing orientation: we explicitly list the basis vectors of one space in the coordinates of some different space.

- The matrix form of representing orientation is useful primarily because it allows us to rotate vectors between coordinate spaces.

- Modern graphics APIs express orientation using matrices.

- We can use matrix multiplication to collapse matrices for nested coordinate spaces into a single matrix.

Summary of Matrix Form 2

- Matrix inversion provides a mechanism for determining the opposite angular displacement.
- Matrices take two to three times as much memory as the other techniques we will learn.
  - This can become significant when storing large numbers of orientations, such as animation data.
- The numbers in a matrix aren't intuitive for humans to work with.
- Not all matrices are valid for describing an orientation.
  - Some matrices contain mirroring or skew.
  - We can end up with a malformed matrix either by getting bad data from an external source, or through matrix creep.

Euler Angles

Euler Angles

- Euler angles are another common method of representing orientation.
  - Euler is pronounced “oiler,” not “yoolur.”
- Named for the famous mathematician who developed them, Leonhard Euler (1707 –1783).

(Image from Wikimedia Commons.)

Euler Angles

- Specify orientation as a series of 3 angular displacements from upright space to object space.
- Which axes? Which order?
  - Need a convention.
- Heading-pitch-bank (Yaw-Pitch-Roll)
  - Heading: rotation about *y* axis (aka "yaw")
  - Pitch: rotation about *x* axis
  - Bank: rotation about *z* axis (aka "roll")

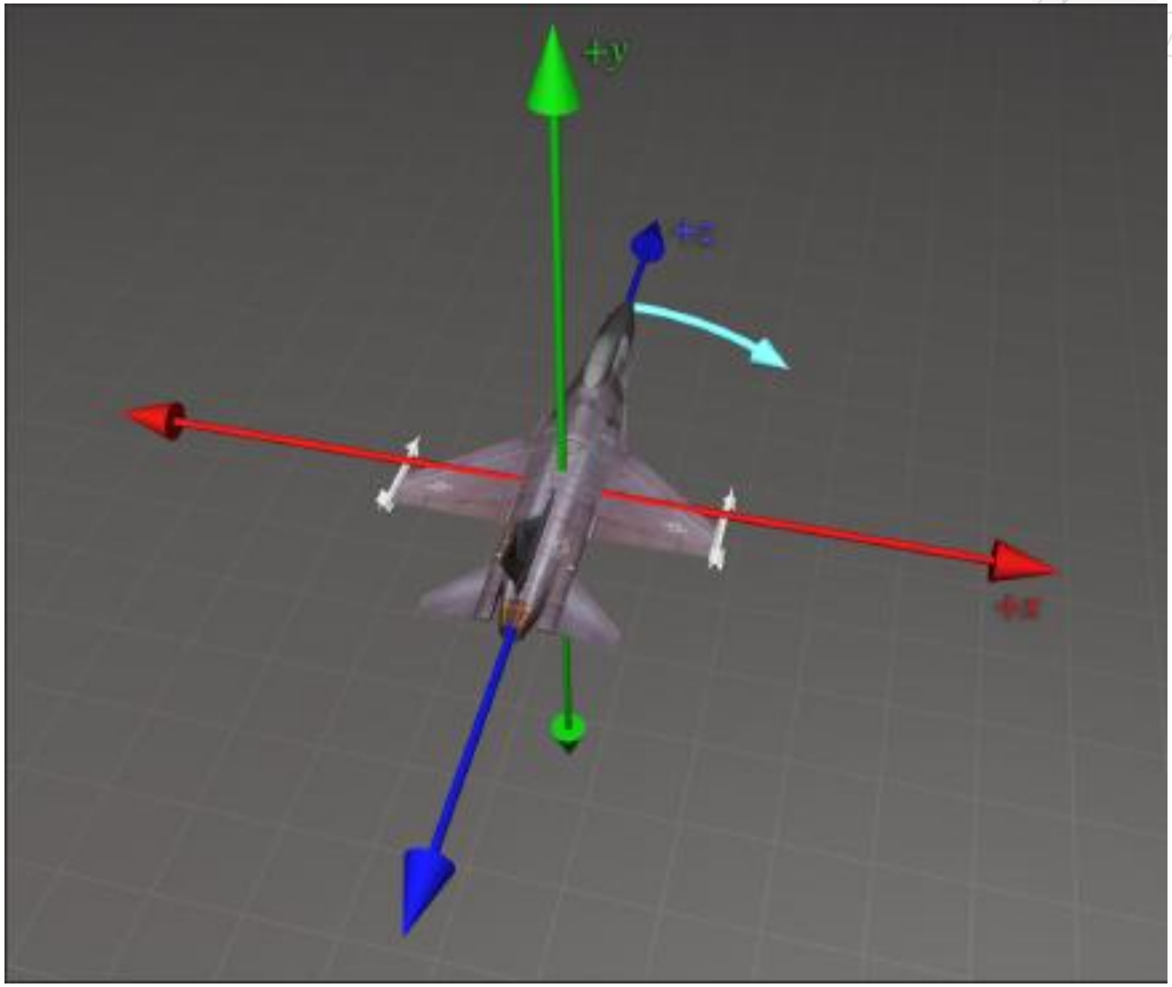
Implementing Euler Angles

- Each game object keeps track of its current heading, pitch, and bank angles (the “Euler angles”).

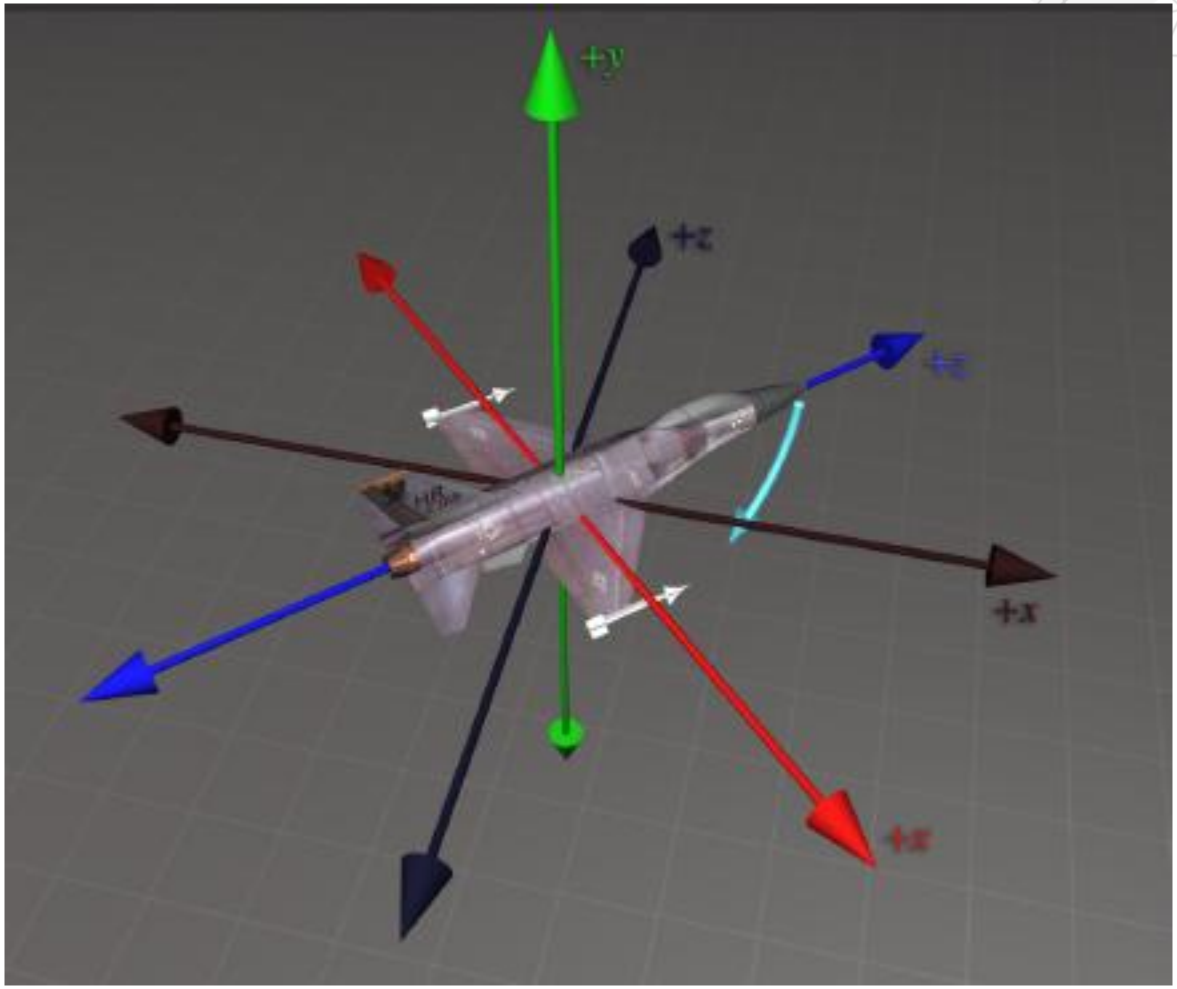
- It also keeps track of its heading, pitch, and bank change rate.

- In each frame, calculate how much the object has changed its heading, pitch, and bank based on the amount of time since the last frame, and add this to the Euler angles.

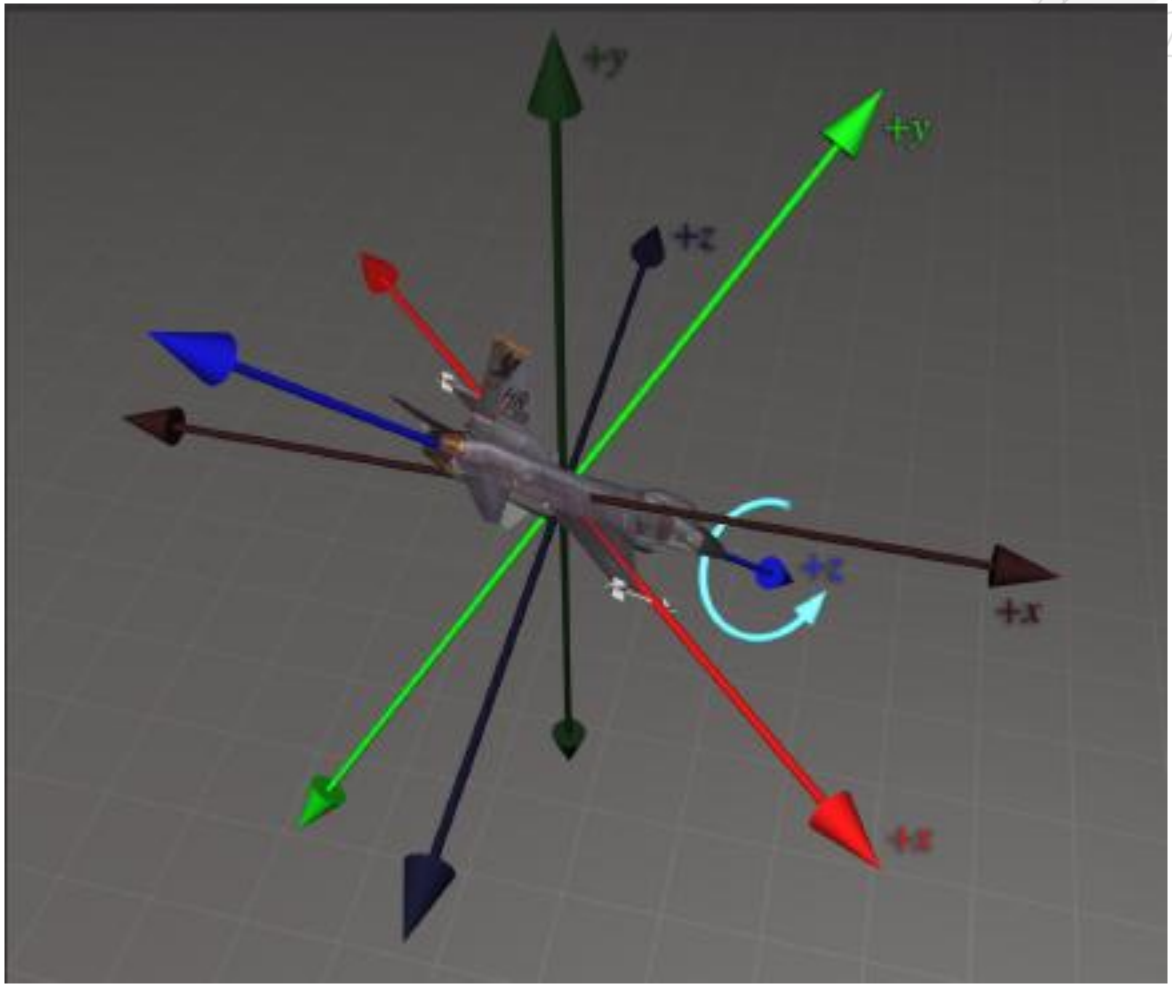
Heading



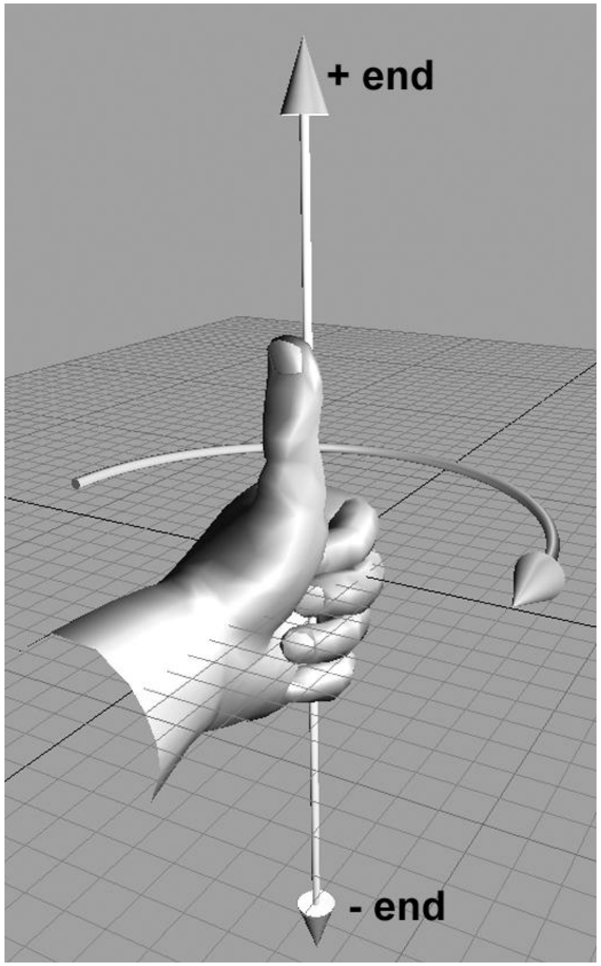
Pitch



Bank



The Sign Matters



- Use the hand rule again.
- Thumb points along positive axis of rotation.
- Fingers curl in direction of positive rotation.

The Order Matters

- Heading is first: it is relative to the upright frame of reference – that is, vertical.

- Pitch is next because it is relative to the horizon. But the *x*-axis may have been moved by the heading change. (Object *x* is no longer the same as upright *x*.)

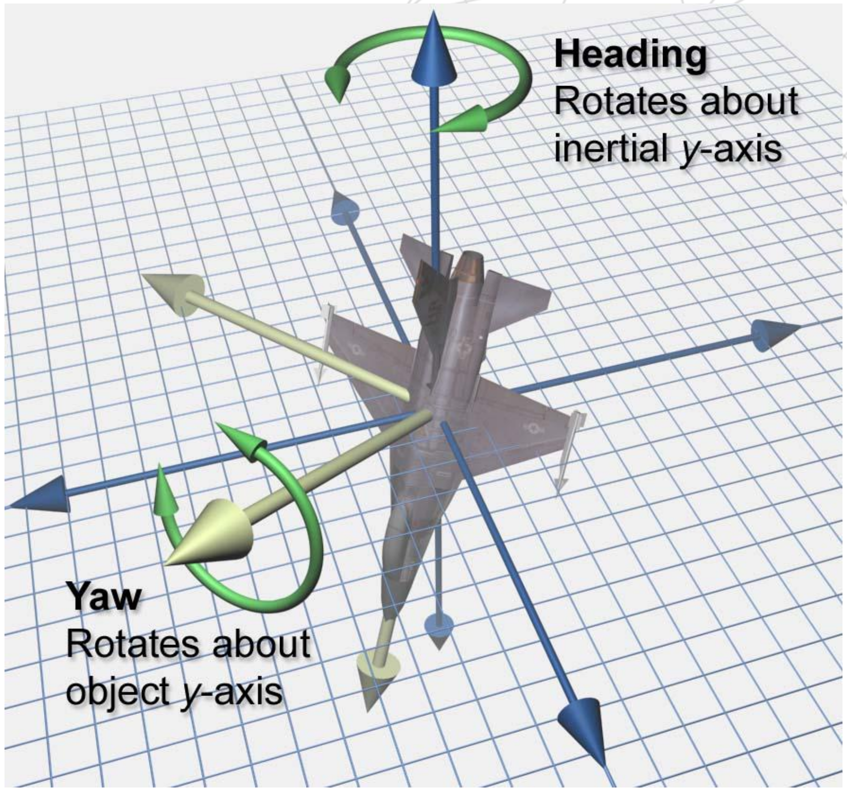
- Bank is last. The *z*-axis may have been moved by the heading and pitch change. (Object *z* is no longer the same as upright *z*.)

Change in name, change in order

- Heading-pitch-bank is often called *yaw-pitch-roll*
  - Heading is yaw, bank is roll.
  - This came from the aerospace industry, where yaw in fact *doesn't* mean heading the way we interpret it.
    - https://en.wikipedia.org/wiki/Aircraft_principal_axes
- Perhaps more interesting is that fact that you will often hear these same three words listed in the opposite order: roll-pitch-yaw.
- As it turns out, there is a perfectly reasonable explanation for this backwards convention: it's the order that we actually do the rotations inside a computer!

The Fixed-Axis System

- In a Euler angle system, the axes of rotation are the body axes, which change after each rotation.
- In a *fixed-axis system*, the axes of rotation are the upright space axes.
- We might visualize Euler angles, inside a computer when rotating vectors from upright space to object space, we will actually use a fixed-axis system.


Heading
Rotates about inertial *y*-axis
Yaw
Rotates about object *y*-axis

Example

- The fixed-axis system and the Euler angle system are equivalent, provided that you take the rotations in the opposite order. Let's say we have a heading (yaw) of *h* and a pitch of *p*.
- According to the Euler angle convention:
  - First do the heading (yaw), rotate about the object space *y*-axis by *h* (yaw).
  - Then pitch, rotate about the object space *x*-axis by *p*.
- Using a fixed-axis scheme we get to this same ending orientation by doing the rotations in the opposite order.
  - First pitch, rotate about the upright *x*-axis by *p*.
  - Then heading, rotate about the upright *y*-axis by *h*.

Pro and cons of Euler Angles

- Pro
  - Easy for humans to use. Really the only option if you want to enter an orientation by hand.
  - Minimal space: 3 numbers per orientation.
    - Save space as the numbers might be more easily compressed.
  - Every set of 3 numbers makes sense – unlike matrices and quaternions.

- Cons
  - Aliasing
  - Gimbal lock
  - Interpolation problems

Aliasing

- There are many ways to represent a single orientation
  - E.g., Pitch down 135° = heading 180°, pitch down 45°, then bank 180°.
- This is called the *aliasing* problem.
- Makes it hard to convert orientations from object to world space.
  - E.g., “Am I facing East?”
- Usual solution: use canonical form

Canonical Euler Angles

- Limit heading (*y*) to ±180°
- Limit pitch (*x*) to ±90°
- Limit bank (*z*) to ±180°

$$-180^\circ < h \leqslant 180^\circ$$
$$-90^\circ \leqslant p \leqslant 90^\circ$$
$$-180^\circ < b \leqslant 180^\circ$$

- Now each orientation has a unique canonical Euler angle.

- Except for one more irritating thing: Gimbal Lock.

Gimbal Lock

- Change heading (*y* axis) by 45°
- Pitch down 90°
- Now if you bank, your object space *z* axis is pointing in world space where your object space *y* axis was before you started your rotations.

- Any rotation about *z* could have been done as a heading change.

- Visual explainers:
  - Robot arm: https://youtu.be/V-Pf0iQEO-k?t=22
  - Axes lock: http://www.youtube.com/watch?v=zc8b2Jo7mno
  - Apollo 13: https://www.youtube.com/watch?v=OmCzZ-D8Wdk

Canonical Euler Angles

- Fix this (i.e. make canonical Euler angles unique) by insisting that if pitch is 90°, then bank must be zero. Put the bank rotation in the heading instead.

$$-180^\circ < h \leqslant 180^\circ$$
$$-90^\circ \leqslant p \leqslant 90^\circ$$
$$-180^\circ < b \leqslant 180^\circ$$
$$\boxed{p = \pm 90^\circ \quad \Rightarrow \quad b = 0}$$

More on Gimbal Lock

- Common misconception: because of Gimbal lock, certain orientations cannot be described using Euler angles.
  - Actually, to describe an orientation, aliasing doesn't pose any problems.

- Any orientation in 3D can be described using Euler angles.
  - That representation is unique within the canonical set.

- Euler angles are always valid.
  - Even if the angles are outside the usual range, we can always agree what orientation is described by the Euler angles.

Aliasing and Gimbal Lock

- What's the fuss about aliasing and Gimbal lock?
- It's about interpolation when animating from one orientation to another.
  - You cannot control if and when the Gimbal lock will happen.
- We wish to interpolate between orientations $\mathbf{R}_0$ and $\mathbf{R}_1$.
- For a given parameter t, $0 \le t \le 1$, we wish to compute an intermediate orientation $\mathbf{R}(t)$ that interpolates smoothly from $\mathbf{R}_0$ to $\mathbf{R}_1$ as t varies from 0 to 1.
  - This is extremely useful for character animation and camera control.

The Dangers of Lerping

- The naive approach to this problem is to apply the standard linear inter**erp**olation formula (*lerp* for short) to each of the three angles independently.

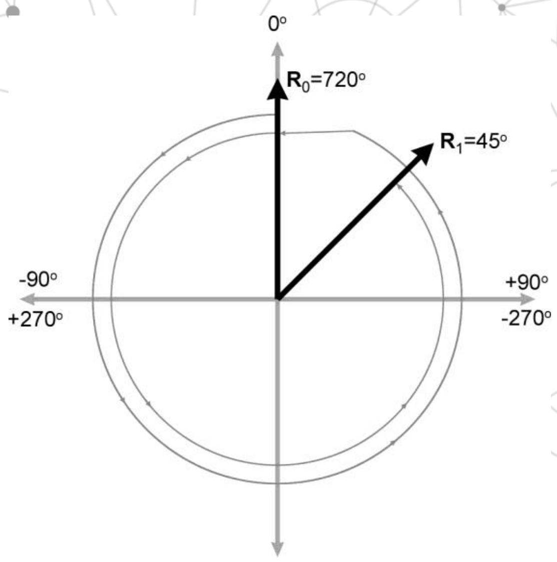
$$\Delta\theta = \theta_1 - \theta_0$$

$$\theta_t = \theta_0 + t\,\Delta\theta$$

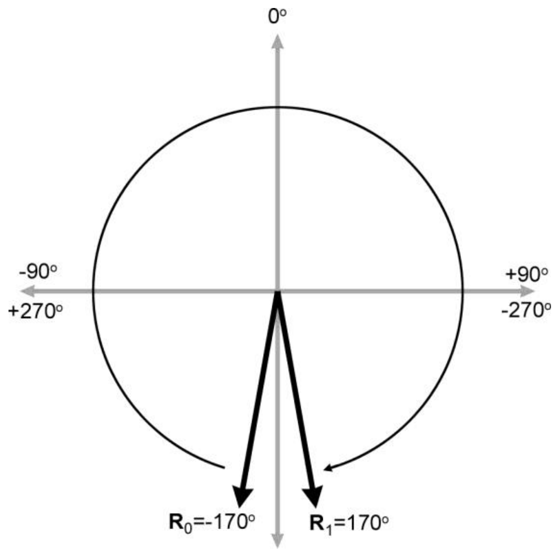
$t,\ 0 \leq t \leq 1$

Problems

- Not good with non-canonical angles.
  - E.g., to interpolate from 720° to 45° in 1° increments would mean turning around twice.



- Not good with canonical Euler Angles.
  - E.g., from –170° to 170°. It goes the "long way round".



Function wrapPi

- Define the following function:

$$\text{wrapPi}(x) = x - 360^\circ \lfloor (x + 180^\circ)/360^\circ \rfloor$$

- Using wrapPi() makes it easy to take the shortest arc when interpolating between two angles.

- We will use wrapPi() in the interpolation formula:

$$\Delta\theta = \text{wrapPi}(\theta_1 - \theta_0)$$
$$\theta_t = \theta_0 + t\,\Delta\theta$$

Another Problem

- Even with these band-aid solutions, you are left with gimbal lock.
- Solution: don’t use Euler angles for interpolation. Use something else, like quaternions.

Summary of Euler Angles 1

- Euler angles store orientation using three angles.
  - These angles are ordered rotations about the three object-space axes.
- The most common system of Euler angles is the heading-pitch-bank system.
  - Heading and pitch tell which way the object is facing, with heading giving a compass reading and pitch measuring the angle of declination. Bank measures the amount of twist.
- In a fixed-axis system, the rotations occur about the upright axes rather than the moving body axes.
  - This system is equivalent to Euler angles provided that we perform the rotations in the opposite order.
- In most situations, Euler angles are more intuitive for humans to work with compared to other methods of representing orientation.

Summary of Euler Angles 2

- Euler angles use the minimum amount of data possible for storing an orientation in 3D
  - They can be more easily compressed than quaternions.
- Euler angles are always valid.
  - Any three numbers have a meaningful interpretation.
- Euler angles suffer from aliasing problems.
  - Due to the cyclic nature of rotation angles, and because the rotations are not completely independent of one another.
- Using canonical Euler angles can simplify many basic queries on Euler angles.
  - Heading and bank are in range –180° to 180°. Pitch is in range –90° to 90°. If pitch is ±90°, then bank is zero.
- Gimbal lock occurs when pitch is ±90°.
  - In this case, one degree of freedom is lost because heading and bank both rotate about the vertical axis.

Summary of Euler Angles 3

- Contrary to popular myth, *any* orientation in 3D can be represented using Euler angles, and we can agree on a unique representation for that orientation within the canonical set.
- The wrapPi function is a very handy tool that simplifies situations where we are dealing with the cyclic nature of angles.
  - Such situation arise frequently in practice.
- Simple forms of aliasing are irritating, but there are workarounds. Gimbal lock is a more fundamental problem and no easy solution exists.

Axis-Angle and Exponential Map

Euler's Rotation Theorem

- Euler’s Rotation Theorem: any 3D angular displacement can be accomplished via a single rotation about a carefully chosen axis.

- For every pair of orientations $\mathbf{R}_1$ and $\mathbf{R}_2$ there exists an axis $\mathbf{n}$ such that we can get from $\mathbf{R}_1$ to $\mathbf{R}_2$ by performing a rotation about $\mathbf{n}$.

- Euler's Rotation Theorem leads to two closely-related methods for describing orientation.
  - *axis-angle form*
  - *exponential map*

Axis-Angle

- Let's say we have chosen a rotation angle θ, and an axis of rotation that passes through the origin and is parallel to the unit vector **n**.
- The two values **n** and θ describe an angular displacement in *axis-angle form*.

Exponential Maps

- Alternatively, since **n** has unit length, without loss of information we can multiply it by θ, yielding the single vector **e** = θ**n**.

- This scheme for describing rotation goes by the name of *exponential map*.
  - The rotation angle can be deduced from the length of **e**, i.e. θ = ||**e**||, and
  - The axis is obtained by normalizing **e**.

- Exponential map has advantages over axis-angle
  - More compact: use 3 numbers instead of 4.
  - It elegantly avoids certain singularities.
  - Has better interpolation and differentiation properties.

Multiples of Angular Displacements

- We can directly obtain a multiple of an angular displacement.
- For example, given a rotation we can get 1/3rd of the rotation, or 2.65 times the rotation
  - In axis-angle form simply by multiplying θ by this amount.
  - With the exponential map just as easily with multiplication.

Exponential Map vs Axis-Angle

- The exponential map gets more use than axis-angle.
  - Its interpolation properties are nicer than Euler angles.
  - Although it does have singularities, they are not as troublesome as Euler angles.
- Usually when one thinks of interpolating rotations one immediately thinks of quaternions, but for some applications exponential map is a viable alternative
  - i.e., for storage of animation data
- The most important and frequent use of the exponential map is for angular *velocity,* not for *displacement*
  - exponential map differentiates nicely.
  - can represent multiple rotations easily.

Aliasing and Singularities

- Like Euler angles, the axis-angle and exponential map forms exhibit aliasing and singularities.
  - Slightly more restricted and benign.
- There is an obvious singularity at the identity orientation, when θ = 0 and any axis may be used.
  - Exponential map nicely tucks this singularity away.
- Another trivial form of aliasing in axis-angle space can be produced by negating both θ and **n**.
  - Exponential map dodges this issue as well, since negating both θ and **n** leaves **e** = θ**n** unchanged!

Other Aliases

- As with Euler angles, adding a multiple of 360° to θ produces an angular displacement that results in the same orientation, in both the axis-angle and the exponential map.

- Not always a shortcoming: for describing angular displacement, this ability to represent such extra rotation is an important and useful property.
  - For example, it's quite important to be able to distinguish between rotation about the *x*-axis at a rate of 720° per second, versus rotation about the same axis at a rate of 1080° per second, even though these displacements result in the same ending orientation if applied for an integral number of seconds.

- It is not possible to capture this distinction in quaternion format.

Quaternions

Real Numbers

- Numbers can be added and multiplied.
  - Real numbers.
  - Visualized as points on a line.
- We need them as the physical world is very well described by real numbers and their arithmetic.
  - “Force is mass times acceleration” is one of the basic laws of motion, and it requires multiplication.
- Addition and multiplication can be nicely used with each other.
- Some properties of the real and rational numbers makes them the foundational structure of math, physics, and all of science and engineering

Complex Numbers (pairs of numbers)

- At some point, we had the idea of using another type of numbers to solve our problems: the complex numbers.
  - Complex numbers are usually explained by introducing a new number $\mathrm{i} = \sqrt{-1}$

- We may also think about a simpler representation: pairs of real numbers.
  - Like (1,0) or (−17,0.29) or ($\sqrt{2}$,π).
  - Points on a plane.

- These numbers have their own algebra.

- Adding pairs of numbers is straightforward. Add the components

$$(2,3) + (4,5) = (6,8)$$

Complex Numbers (pairs of numbers)

- Multiplication of pairs is less straightforward:

  $$(a,b)\times(c,d)=(ac-bd,\ ad+bc)$$

- Think of (a,b) as a new kind of number a+bi
  - Where i is a symbol with property $i^2 = -1$
  - i.e., its square is −1

- Multiplying a+bi and c+di is nothing more than expanding a product using the distributive law.

  $$(a + ib)\,(c + id) = ac + iad + ibc + i^2bd =$$

  $$= (ac - bd) + i(ad + bc)$$

  $-$ Bd because $i^2 = -1$

Complex Numbers (triples of numbers)

- Sir William Rowan Hamilton set out to find a way of multiplying triplets instead of pairs.

- The physical world is 3D, so we need triplets right?
  - Single numbers represent points on a line
  - Pairs of numbers represent points in the plane
  - Triplets (x,y,z) of numbers represent points in space

- Adding triplets is still done as before, one by one: (1,2,3)+(10,100,1000)=(11,102,1003)

Sir William Rowan Hamilton

- How to multiply them?
- Hamilton, was looking for a formula similar to (ac−bd, ad+bc)
    - Years later, he recalled that period in a letter to his son: {Every morning... on my coming down for breakfast, your (then) little brother William Edwin, and yourself, used to ask me "Well, Papa, can you multiply triplets?". Whereto I was always obliged to reply, with a sad shake of the head: "No, I can only add and subtract them."}

- Hamilton was bound to fail.
  - An algebraic structure of that type does not exist for triplets.
    - For geometric (i.e., topological) reasons.

The birth of Quaternions

- Hamilton did not give up, and one Monday morning, on October 16, 1843…

- On that Monday morning Hamilton took a walk with his wife along the Royal Canal in Dublin, when all of a sudden he saw the answer:

- Forget triplets. Multiply quadruplets instead.
  - Take four real numbers as your basic object, and everything will work like magic.
  - Hamilton literally scribbled the formulas on a bridge, and the event is commemorated in a plaque to this day.

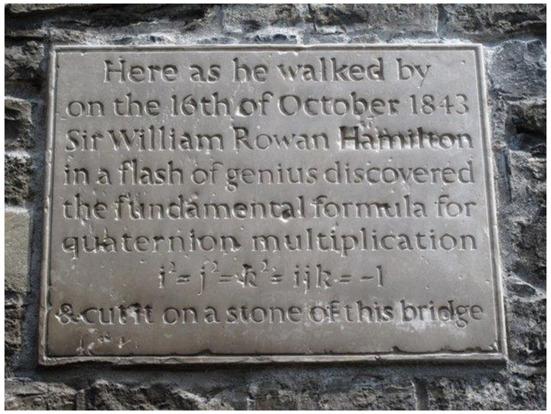
- Multiplication of quadruplets is beautiful, clean, useful, but it is not commutative.

$$A \times B \neq B \times A$$

Quaternions intuition

- Instead of just a single number i whose square is −1, introduce three of them.

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



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

- They multiply among themselves as follows:

$$ij=k, \; jk=i, \; ki=j$$

- However, if we flip things around:

$$ji=-k, \; kj=-i, \; ik=-j$$

Quaternions can be multiplied

- With these rules, we can multiply (a,b,c,d) and (e,f,g,h) by writing them like so:

  (a+bi+cj+dk)(e+fi+gj+hk)=…

  - expand the product as you usually would, remembering the special rules for ijk.

- Observe that you can't do it without the a part, since multiplying i or j or k by themselves yields −1, an ordinary real number.

- Quaternion a+bi+cj+dk has a "scalar" part a and a 3-dimensional "vector" part bi+cj+dk.
  - Like a complex number a+bi has a so-called "real part" a and "imaginary part" bi.

Quaternions: Vector-Scalar Notation

Instead of the complex number notation:

$$q = w + xi + yj + zk.$$

use vector-scalar notation:

$$\mathbf{q} = [\, w\ \mathbf{v} \,],$$

where *w* is a scalar and **v** a vector. Alternatively:

$$\mathbf{q} = [\, w\ (\, x\ y\ z \,)\,]$$

(it’s the same *w*, *x*, *y*, *z*).

Quaternions: 4D Space

- Hamilton himself thought of quaternions as 4D vectors [*w*, *x*, *y*, *z*].
- However, quaternions are **not** the same as vectors in homogenous 4D space [*x*, *y*, *z*, *w*].
- Careful! His *w* is **not** the same as the *w* in homogenous 4D space.

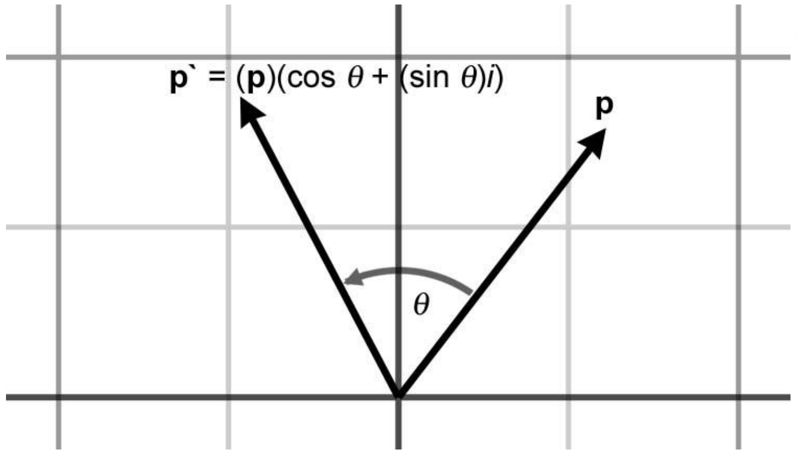
2D and Complex Numbers

- Instead of thinking of a point (*x*, *y*) in 2D space as two real numbers, think of it as a single complex number *x* + *yi*.
- A rotation by angle θ can be **also** be represented as a complex number

$$\cos\theta + i\sin\theta$$

- Complex numbers are doing double duty here as both points and rotations.

Complex Multiplication



A rotation (represented as a complex number) can be applied to a point (also represented as a complex number) using multiplication of complex numbers.

$$(x + yi)\,(\cos\theta + i\sin\theta)$$

$$= (x\cos\theta - y\sin\theta) + i\,(x\sin\theta + y\cos\theta)$$

What's Going On Here?

Remember the 2D rotation matrix. The imaginary part captures the negative sign on the sine (so to speak).

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

$$\underbrace{(x + yi)}_{\text{point } (x,y)} \underbrace{(\cos\theta + i\sin\theta)}_{\text{rotation}}$$

$$= (x\cos\theta - y\sin\theta) + i\,(x\sin\theta + y\cos\theta)$$

rotated point (x',y')

What Do Imaginary Numbers Buy Us?

The imaginary part seems to have two roles.

1. It allows us to “jam together” the *x* and *y* part from a vector [*x*, *y*] into a single number *x* + *yi*. The imaginary part *i* keeps the *x* and *y* parts separate.

2. It gets us the sign change on the sin θ that we saw in rotation matrices.

Points as Quaternions

- Represent 3D point ($x\ y\ z$) as the quaternion

$$\mathbf{p} = [\ 0\ (\ x\ y\ z\ )\ ].$$

- or equivalently in complex number notation:

$$xi + yj + zk.$$

  - Games generally don’t implement the 0 value for *w*.

Rotations as Quaternions

- The scalar part "kind of" represents the angle.
- The vector part "kind of" represents the axis.

- Rotation by an angle θ around unit axis **n** is represented by the quaternion:

$$\mathbf{Q} = [\ \cos \theta/2 \quad \sin \theta/2\ \mathbf{n}\ ]$$
$$= [\ \cos \theta/2 \quad \sin \theta/2\ (\mathbf{n}_x\ \mathbf{n}_y\ \mathbf{n}_z)]$$

Quaternion Negation

- If **q** = [w (x y z)] = [w **v**] is a quaternion, define its negation to be:

$$-\mathbf{q} = -[w\ (x\ y\ z)] = [-w\ (-x\ -y\ -z)]$$
$$= -[w\ \mathbf{v}] = [-w\ -\mathbf{v}].$$

- Surprisingly, **q** = –**q**. The quaternions **q** and –**q** describe the same angular displacement.

- Any angular displacement in 3D has exactly two distinct representations in quaternion format, and they are negatives of each other.
  - If we add 360° to θ, it doesn't change the angular displacement represented by **q**, but it negates all four components of **q**.

Quaternion Magnitude

- The magnitude of a quaternion is computed as:

$$\begin{aligned}\|\mathbf{q}\| &= \left\|\begin{bmatrix} w & (x \quad y \quad z)\end{bmatrix}\right\| = \sqrt{w^2 + x^2 + y^2 + z^2} \\ &= \left\|\begin{bmatrix} w & \mathbf{v}\end{bmatrix}\right\| = \sqrt{w^2 + \|\mathbf{v}\|^2}\end{aligned}$$

- The magnitude of a rotation quaternion is 1.

$$\begin{aligned}\|\mathbf{q}\| &= \left\|\begin{bmatrix} w & \mathbf{v}\end{bmatrix}\right\| = \sqrt{w^2 + \|\mathbf{v}\|^2} \\ &= \sqrt{\cos^2(\theta/2) + (\sin(\theta/2)\|\hat{\mathbf{n}}\|)^2} && (\text{Substituting using } \theta \text{ and } \hat{\mathbf{n}}) \\ &= \sqrt{\cos^2(\theta/2) + \sin^2(\theta/2)\|\hat{\mathbf{n}}\|^2} \\ &= \sqrt{\cos^2(\theta/2) + \sin^2(\theta/2)\ 1} && (\hat{\mathbf{n}} \text{ is a unit vector.}) \\ &= \sqrt{1} && (\sin^2 x + \cos^2 x = 1) \\ &= 1\end{aligned}$$

Conjugate and Inverse

The *conjugate* of a quaternion is obtained by reversing the vector part.

$$\mathbf{q}^* = [\, w\ \mathbf{v}\,]^* = [w\ \ -\mathbf{v}\,].$$

The inverse of a quaternion is its conjugate divided by its magnitude.

$$\mathbf{q}^{-1} = \mathbf{q}^*/\|\mathbf{q}\|.$$

For unit quaternions, in particular rotation quaternions, conjugate is the same as inverse

$$[\, w\ \mathbf{v}\,]^* = [\, w\ \mathbf{v}\,]^{-1}.$$

The inverse of a rotation quaternion is a rotation in the opposite direction.

Quaternion Multiplication

$$
\begin{aligned}
\mathbf{q}_1\mathbf{q}_2 &= \begin{bmatrix} w_1 & \begin{pmatrix} x_1 & y_1 & z_1 \end{pmatrix} \end{bmatrix} \begin{bmatrix} w_2 & \begin{pmatrix} x_2 & y_2 & z_2 \end{pmatrix} \end{bmatrix} \\
&= \begin{bmatrix} w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2 \\ \begin{pmatrix} w_1x_2 + x_1w_2 + y_1z_2 - z_1y_2 \\ w_1y_2 + y_1w_2 + z_1x_2 - x_1z_2 \\ w_1z_2 + z_1w_2 + x_1y_2 - y_1x_2 \end{pmatrix} \end{bmatrix} \\
&= \begin{bmatrix} w_1 & \mathbf{v}_1 \end{bmatrix} \begin{bmatrix} w_2 & \mathbf{v}_2 \end{bmatrix} \\
&= \begin{bmatrix} w_1w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2 & w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2 \end{bmatrix}
\end{aligned}
$$

Facts About Quaternion Multiplication

- It is associative but not commutative.

$$\mathbf{q}(\mathbf{rs}) = (\mathbf{qr})\mathbf{s} \qquad \mathbf{qr} \neq \mathbf{rq}$$

- Multiplicative identity is $[1\ \mathbf{0}] = [1\ (\ 0\ 0\ 0\ )]$

- $\|\mathbf{qr}\| = \|\mathbf{q}\|\ \|\mathbf{r}\|$

- $(\mathbf{ab})^{-1} = \mathbf{b}^{-1}\mathbf{a}^{-1}$, and in general (just like matrices)

$$(\mathbf{q}_1\mathbf{q}_2 \ldots \mathbf{q}_{n-1}\mathbf{q}_n)^{-1} = \mathbf{q}_n^{-1}\mathbf{q}_{n-1}^{-1} \ldots \mathbf{q}_2^{-1}\mathbf{q}_1^{-1}$$

Applying Rotations to Points

- To apply a rotation quaternion **q** to a 3D point *p*, use quaternion multiplication (thinking of *p* for a moment as the quaternion [1 *p*]):

$$\mathbf{q}p\mathbf{q}^{-1}.$$

- Examine what happens when multiple rotations are applied to a vector. Rotate the vector **p** by the quaternion **a**, and then rotate that result by another quaternion **b**.

$$\begin{aligned}\mathbf{p}' &= \mathbf{b}(\mathbf{a}\mathbf{p}\mathbf{a}^{-1})\mathbf{b}^{-1}\\ &= (\mathbf{b}\mathbf{a})\mathbf{p}(\mathbf{a}^{-1}\mathbf{b}^{-1})\\ &= (\mathbf{b}\mathbf{a})\mathbf{p}(\mathbf{b}\mathbf{a})^{-1}.\end{aligned}$$

Concatenating Rotations

- Quaternion multiplication can be used to concatenate multiple rotations
  - Rotating by **a** and then by **b** is equivalent to performing a single rotation by the quaternion product **ba**.

- Different from matrix multiplication.
  - With matrix multiplication, we use row vectors (on the left) and the concatenated rotations read left-to-right in the order of transformation.
  - With quaternions, concatenation of multiple rotations will always read from right to left.

Order of Multiplication

- This means that to apply a rotation quaternion **q** followed by a rotation quaternion **r**, we apply the product quaternion **qr**:

$$\mathbf{r}^{-1}(\mathbf{q}^{-1} p\, \mathbf{q})\mathbf{r} = (\mathbf{qr})^{-1} p\, (\mathbf{qr})$$

- This is cool because quaternion multiplication is done the same order as the transformations.

Quaternion Difference

- Let **a** and **b** be quaternions representing two orientations.
- The quaternion **d** that takes orientation **a** to orientation **b** is called the *quaternion difference* between **a** and **b**.
- Want **ad** = **b**.
- What is **d**?

$$\mathbf{d} = \mathbf{a}^{-1}\mathbf{b}$$

- Why?

$$\mathbf{ad} = \mathbf{a}(\mathbf{a}^{-1}\mathbf{b}) = (\mathbf{a}\mathbf{a}^{-1})\mathbf{b} = [\ 1\ \mathbf{0}\ ]\ \mathbf{b} = \mathbf{b}$$

Quaternion Dot Product

- Similar to vector dot product:

$$\mathbf{q}_1.\mathbf{q}_2 = [\, w_1\ \mathbf{v}_1 \,] . [\, w_2\ \mathbf{v}_2 \,] = w_1\, w_2 + \mathbf{v}_1.\mathbf{v}_2$$

- If $\mathbf{v}_1 = [\, x_1\ y_1\ z_1 \,]$ and $\mathbf{v}_2 = [\, x_2\ y_2\ z_2 \,]$, then:

$$\mathbf{q}_1.\mathbf{q}_2 = w_1\, w_2 + x_1\, x_2 + y_1\, y_2 + z_1\, z_2$$

- Geometric interpretation: the larger the absolute value of $\mathbf{q}_1.\mathbf{q}_2$, the more similar their orientations are.

Quaternion Exponentiation

- Quaternion exponentiation is useful because it allows us to extract a fraction of an angular displacement.

- If $\mathbf{q}$ is a quaternion and t is a scalar, define

$$\mathbf{q}^t = \exp(t \log \mathbf{q}).$$

- As t varies from 0 to 1, the quaternion $\mathbf{q}^t$ varies from $[1, \mathbf{0}]$ to $\mathbf{q}$.

- For example, $\mathbf{q}^{1/3}$ is a quaternion that represents 1/3rd of the angular displacement represented by the quaternion $\mathbf{q}$.

Facts About Quaternion Exponentiation

- Exponents t outside the range $0 \le t \le 1$ behave mostly as expected, with one major caveat.
- For example, $\mathbf{q}^2$ represents twice the angular displacement as $\mathbf{q}$.
- If $\mathbf{q}$ represents a clockwise rotation of 30° about the x-axis, then $\mathbf{q}^2$ represents a clockwise rotation of 60° about the x-axis, and $\mathbf{q}^{-1/3}$ represents a counterclockwise rotation of 10° about the x-axis.
- Notice that $\mathbf{q}^{-1}$ yields the quaternion inverse.

The Caveat

- The caveat we mentioned is this: a quaternion represents angular displacements using the shortest arc.
  - Multiple spins cannot be represented.
- For example, if **q** represents a clockwise rotation of 30° about the *x*-axis, then $\mathbf{q}^8$ is not a 240° clockwise rotation about the x-axis as expected; it is a 120° counterclockwise rotation.
- Of course, rotating 240° in one direction produces the same end result as rotating 120° in the opposite direction, and this is the point: *quaternions only really capture the end result.*

In Consequence

- If further operations on this quaternion were performed, things will not behave as expected. For example, $(\mathbf{q}^8)^{1/2}$ is not $\mathbf{q}^4$, as we would intuitively expect.
  - In general, many of the algebraic identities concerning exponentiation of scalars, such as $(a^s)^t = a^{st}$, do not apply to quaternions.
- In some situations, we do care about the total amount of rotation, not just the end result.
- The most important example is that of angular velocity.
  - Quaternions are not the correct tool for this job; use the exponential map (or axis-angle format)

Quaternion Interpolation

- The *raison d'etre* of quaternions in games and graphics today is an operation known as *slerp*, which stands for **s**pherical **l**inear int**erp**olation.

- Slerp allows us to smoothly interpolate between two orientations while avoiding all the problems that plagued interpolation of Euler angles.

- Slerp is a ternary operator, meaning it accepts three operands, 2 quaternions and a scalar.

The Slerp Function

- The first two operands the two quaternions $\mathbf{q}_0$ and $\mathbf{q}_1$ between which we wish to interpolate.
- The third operand is the interpolation parameter, a real number t such that $0 \le t \le 1$.
- As t varies from 0 to 1, the slerp function

$$\text{slerp}(\mathbf{q}_0, \mathbf{q}_1, t)$$

- will return an orientation that interpolates from $\mathbf{q}_0$ to $\mathbf{q}_1$ by fraction t.

Scalar Linear Interpolation

To interpolate between two scalar values a_0 and a_1, use the standard linear interpolation formula:

$$\Delta a = a_1 - a_0$$

$$\text{lerp}(a_0, a_1, t) = a_0 + t.\Delta a$$

This requires three basic steps:

1. Compute the difference between the two values
2. Take a fraction of this difference
3. Take the first value and adjust it by this fraction of the difference.

- Putting these steps together,

$$\text{slerp}(\mathbf{q}_0, \mathbf{q}_1, t) = (\mathbf{q}_1\mathbf{q}_0^{-1})^t\, \mathbf{q}_0.$$

- This is how slerp is computed in theory.
  - In practice, a more efficient technique is used.

Advantages of Quaternions

- **Smooth interpolation**. The interpolation provided by slerp provides smooth interpolation between orientations.
  - No other representation method provides for smooth interpolation.
- **Fast concatenation and inversion of angular displacements.** We can concatenate a sequence of angular displacements into a single angular displacement using the quaternion cross product.
  - The same operation using matrices involves more scalar operations, though which one is actually faster on a given architectures is not so clean-cut: SIMD vector operations can make very quick work of matrix multiplication.
  - Quaternion conjugate provides a way to compute the opposite angular displacement very efficiently. This can be done by transposing a rotation matrix, but is not easy with Euler angles.
- **Fast conversion to and from matrix form**. Quaternions can be converted to and from matrix form a bit faster than Euler angles.
- **Only four numbers**. Since a quaternion contains four scalar values, it is considerably more economical than a matrix, which uses nine numbers.
  - However, it still is 33% larger than Euler angles.

Disadvantages of Quaternions

However, these advantages do come at some cost.

- **Slightly bigger than Euler angles**. That one additional number may not seem like much, but an extra 33% can make a difference when large amounts of angular displacements are needed, for example, when storing animation data.
  - Moreover, the values inside a quaternion are not evenly spaced from –1 to 1 so the component values do not interpolate smoothly even if the orientation does.

- **Can become invalid**. This can happen either through bad input data, or from accumulated floating point round off error.

- **Difficult for humans to work with**. Of the three representation methods, quaternions are the most difficult for humans to work with directly.

Comparison of Methods

Comparison of Methods 1

Rotating points between coordinate spaces (object and upright)

- **Matrix**: Possible, and often highly optimized by vector instructions.
- **Euler Angles**: Impossible (must convert to rotation matrix)
- **Exponential Map**: Impossible (must convert to rotation matrix)
- **Quaternion**: On a chalkboard, yes. Practically, in a computer, not really. You might as well convert to rotation matrix.

Concatenation of multiple rotations

- **Matrix**: Possible and can often be highly optimized by vector instructions. Watch out for matrix creep.
- **Euler Angles**: Impossible.
- **Exponential Map**: Impossible.
- **Quaternion**: Possible. Fewer scalar operations than matrix multiplication but might not as easy to take advantage of SIMD instructions. Watch out for error creep.

Comparison of Methods 2

Inversion of rotations

- **Matrix**: Easy and fast, using matrix transpose
- **Euler Angles**: Not easy.
- **Exponential Map**: Easy and fast, using vector negation
- **Quaternion**: Easy and fast, using quaternion conjugate

Interpolation

- **Matrix**: Extremely problematic
- **Euler Angles**: Possible, but Gimbal lock causes quirkiness
- **Exponential Map**: Possible, with some singularities, but not as troublesome as Euler angles.
- **Quaternion**: Slerp provides smooth interpolation

Comparison of Methods 3

Direct human interpretation

- **Matrix**: Difficult
- **Euler Angles**: Easiest
- **Exponential Map**: Very difficult
- **Quaternion**: Very difficult

Storing in a memory or in a file

- **Matrix**: Nine numbers
- **Euler Angles**: Three numbers that can be easily quantized
- **Exponential Map**: Three numbers that can be easily quantized
- **Quaternion**: 4 numbers that don't quantize well; can be reduced to 3 by assuming unit length and that 4th component is nonnegative.

Comparison of Methods 4

Unique representation for a given rotation

- **Matrix**: Yes
- **Euler Angles**: No, due to aliasing
- **Exponential Map**: No, due to aliasing; less complicated than Euler angles.
- **Quaternion**: Exactly two distinct representations for any angular displacement, and they are negatives of each other

Possible to become invalid

- **Matrix**: Six degrees of redundancy inherent in orthogonal matrix. Matrix creep can occur.
- **Euler Angles**: Any three numbers can be interpreted unambiguously
- **Exponential Map**: Any three numbers can be interpreted unambiguously
- **Quaternion**: Error creep can occur

How to Choose Representation 1

- Euler angles are easiest for humans to work with. Using Euler angles greatly simplifies human interaction when specifying the orientation of objects in the world.

- This includes direct keyboard entry of an orientation, specifying orientations directly in the code (e.g. positioning the camera for rendering).

- This advantage should not be underestimated.
  - Don't sacrifice ease of use in the name of optimization until you are certain that your optimization will make a difference in practice.

How to Choose Representation 2

- Matrix form must eventually be used if vector coordinate space transformations are needed.
- You can store the orientation using another format and then generate a rotation matrix when you need it.
  - Use Euler angles or a Quaternion

How to Choose Representation 3

- For storage of large numbers of orientations (e.g. animation data), Euler angles, exponential maps, and quaternions offer various tradeoffs.

- In general, the components of Euler angles and exponential maps quantize better than quaternions.

How to Choose Representation 3

- Reliable quality interpolation can only be accomplished using quaternions.

- You can always convert to quaternions, perform the interpolation, and then convert back to the original form.

- Direct interpolation using exponential maps might be a viable alternative in some cases, since the points of singularity are at very extreme orientations and are often easily avoided in practice.

How to Choose Representation 4

- For angular velocity or any other situation where extra spins over and above 360° need to be represented, use the exponential map or axis-angle.

Conversion Algorithms

- At some point you may need to convert between the different representations.

- The most used combinations may include:
  1. Euler angles to rotation matrix.
  2. Rotation matrix to Euler angles.
  3. Quaternion to matrix.
  4. Matrix to quaternion.
  5. Euler angles to quaternion.
  6. Quaternion to Euler angles.

- Some are simpler than others, usually you will need to apply dedicated algorithms.

- Converting needs time. When choosing the representations, consider advantages against the time you will spend converting back and forth.