Computer Graphics

03. Rotations in 3D (part 2)

Dr Joan Llobera – joanllobera@enti.cat

ESCOLA DE NOVES TECNOLOGIES INTERACTIVES

centre adscrit a: UNIVERSITAT DE BARCELONA

Summary last class on rotations:

- Rotations in 2D
  - Angle
  - Matrix
- Rotations in 3D
  - Euler Angles
  - Yaw-Pitch-Roll
  - Axis Angle
  - 3x3 Matrix

Today

- Reminder rotations in 2D
  - Angle
  - Matrix
  - Complex Numbers
- Introduce New method for Rotations in 3D
  - Euler Angles
  - Yaw-Pitch-Roll
  - Axis Angle
  - 3x3 Matrix
  - Quaternions

Rotations

- With complex numbers
- With quaternions

We want to have:

- Compact representation
- Simple calculation
- Robust composition
- Robust interpolation

Reminder on Complex Numbers

Definition:

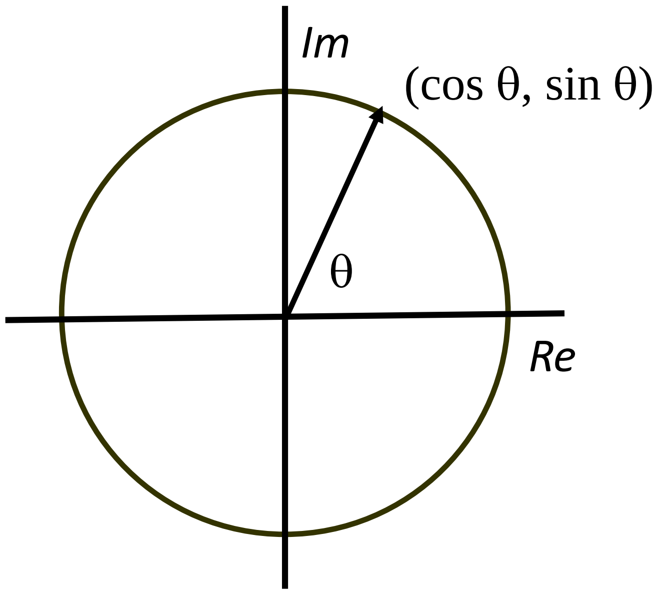
$$z = a + bi$$

with

$$i^2 = -1$$

- Complex numbers are a good compact representation of movements on a plane (2 Values)
- If normalized ($a^2 + b^2 = 1$), can use these to represent 2D rotation

Unit circle on complex plane



Euler Formula
(proof from Taylor Expansions)

$$e^{i\theta} = \cos\theta + i\sin\theta$$

Polar form of a complex number:

$$a + bi = Ae^{i\theta}$$

Euler Identity

You may have seen this:

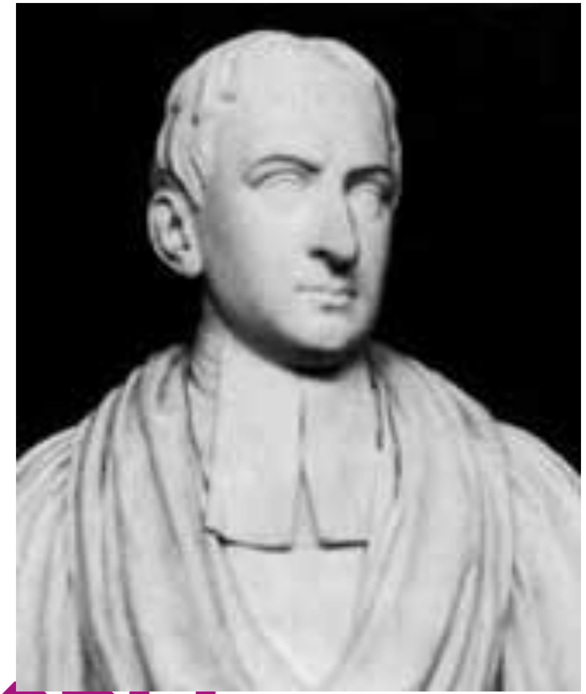
$$e^{\pi\mathbf{i}} + 1 = 0$$

It falls out from:

$$\begin{aligned} 0 &= e^{\pi\mathbf{i}} + 1 \\ &= \cos\pi + \mathbf{i}\sin\pi + 1 \\ &= -1 + \mathbf{i}(0) + 1 \\ &= 0 \end{aligned}$$

Who came up with this?

Roger Cotes in 1714 (sculpture by Scheemakers)



Euler in 1748 (painting by Handmann)

Interpretation on Plane

- Caspar Wessel (1799)
- Jean Robert Argand (1806)
- Made “popular” around 1814

Operations

Conjugate:

(a+bi)* = a -bi

Addition:

(a+bi)+(c+di) = (a+c)+ (b+d)i

Product:

(a+bi) (c+di)= (ac - bd) + (ad +bc)i

$$Ae^{i\theta} * Be^{i\varphi} = ABe^{i(\theta+\varphi)}$$

Conclusion

A multiplication by a complex number of modulus 1 can be seen, in geometrical terms, as a rotation on the plane

Outline

Can we achieve simple and efficient maths for all three?

- Concatenation
- Interpolation
- Rotation

Outline:

- Complex numbers

  (good rotations in 2D)
- Quaternions

  (good rotations in 3D)

And how to go from there to usual space

In addition:

- Review on dot and cross product

Rotations

- With complex numbers (in 2D)
- With quaternions (in 3D)

We want to have:

- Compact representation
- Simple calculation
- Robust composition
- Robust interpolation

What is a Quaternion?

Created as extension to complex numbers

$$a + b\mathbf{i}$$

becomes

$$w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$

Can represent as coordinates

$$(w, x, y, z)$$

Or scalar/vector pair

$$(w, \mathbf{v})$$

Discovery

October 16, 1843

W. R. Hamilton

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

Definition

3 interlinked imaginary values

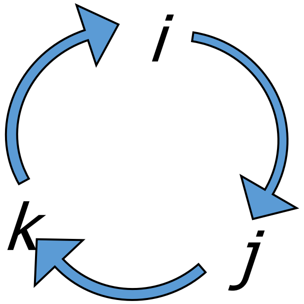
$$q = q_0 + q_1 i + q_2 j + q_3 k = q_0 + \vec{q}$$

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

$$ij = -ji = k$$

$$jk = -kj = i$$

$$ki = -ik = j$$



Operations

- Addition

$$p = p_0 + p_1 i + p_2 j + p_3 k$$

$$q = q_0 + q_1 i + q_2 j + q_3 k$$

$$p + q \equiv (p_0 + q_0) + (p_1 + q_1)i + (p_2 + q_2)j + (p_3 + q_3)k$$

Operations

- Multiplication

$$(p_o + p_1 i + p_2 j + p_3 k)(q_o + q_1 i + q_2 j + q_3 k)$$

Remarks:

- Compare with complex numbers

(a+bi)(c+di)= (ac - bd) + (ad +bc)i

Properties:

- Associative
- Non-commutative

Operations

- Conjugate

$$q^* \equiv q_0 - \vec{q} \qquad (pq)^* \equiv q^* p^*$$

- Modulus (length)

$$|q| = \sqrt{q^* q} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

- Identity quaternion

Remarks:

- Idem to complex (but with 3 imaginary numbers)

Exercise 1

Given:

$$q_1 = (\frac{\sqrt{2}}{2}, 0, 0, \frac{\sqrt{2}}{2})$$
$$q_2 = (\frac{\sqrt{2}}{2}, 0, \frac{\sqrt{2}}{2}, 0)$$

Calculate the products

$$q_1 q_2$$

$$q_2 q_1$$

Applications of Quaternions

Used to represent rotations and orientations of objects in three-dimensional space in:

- Computer graphics
- Control theory
- Signal processing
- Attitude controls
- Physics
- Orbital mechanics
- Quantum Computing

What is a Rotation Quaternion?

Normalized quaternion
is a rotation representation

- If not unitary, it is not a rotation (scaling)
- Normalizing avoids errors due to floating point rounding

- To normalize, multiply by

$$1/\sqrt{x^2 + y^2 + z^2 + w^2}$$

Why *4* values?

One way to think of it:

2D rotation ->

One degree of freedom

Normalized complex number ->

One degree of freedom

3D rotation ->

Three degrees of freedom

Normalized quaternion ->

Three degrees of freedom

How does a Quaternion relate to a Rotation?

Normalized quat (w, x, y, z)

w represents angle of rotation θ

$w = \cos(\theta/2)$

x, y, z form <u>normalized</u> rotation axis $\mathbf{r}$

$(x\ y\ z) = \mathbf{v} = \sin(\theta/2)\cdot\mathbf{r}$

<u>It's a modified axis-angle!</u>

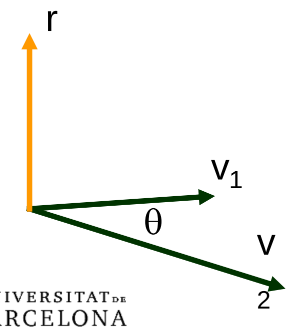
Quaternion as rotations

Have vector $\mathbf{v}_1$, want to rotate to $\mathbf{v}_2$

Need rotation vector $\mathbf{r}$, angle θ

Plug into previous formula

$$\theta = \operatorname{acos}(\hat{\mathbf{v}}_1 \bullet \hat{\mathbf{v}}_2)$$
$$\mathbf{r} = \mathbf{v}_1 \times \mathbf{v}_2$$



Common trick – originally from Game Gems 1 (Stan Melax)

Use trig identities to avoid acos

- Normalize $\mathbf{v}_1$, $\mathbf{v}_2$

$$\mathbf{r} = \hat{\mathbf{v}}_1 \times \hat{\mathbf{v}}_2 \qquad s = \sqrt{2(1 + \hat{\mathbf{v}}_1 \bullet \hat{\mathbf{v}}_2)}$$

Build quat

$$\mathbf{q} = (2s, \mathbf{r} / s)$$

- More stable when $\mathbf{v}_1$, $\mathbf{v}_2$ near parallel

ESCOLA DE NOVES TECNOLOGIES INTERACTIVES

UNIVERSITAT DE BARCELONA

Exercise 1 (revisited)

We did:

Given:

$$q_1 = (\frac{\sqrt{2}}{2}, 0, 0, \frac{\sqrt{2}}{2})$$
$$q_2 = (\frac{\sqrt{2}}{2}, 0, \frac{\sqrt{2}}{2}, 0)$$

Calculate the products

$$q_1 q_2$$

$$q_2 q_1$$

How do you interpret these in terms of rotations?

Example

To rotate 90° around z-axis:

$$w = \cos(45^\circ) = \sqrt{2}/2$$

$$x = 0 \cdot \sin(45^\circ) = 0$$

$$y = 0 \cdot \sin(45^\circ) = 0$$

$$z = 1 \cdot \sin(45^\circ) = \sqrt{2}/2$$

$$\mathbf{q} = (\sqrt{2}/2, 0, 0, \sqrt{2}/2)$$

Exercise 1 (solved in RStudio)

```
rm(list = ls())
install.packages("rotations")
library(ggplot2);
library(rotations);

Q1 <- as.Q4(c(sqrt(2)/2,0,0,sqrt(2)/2))
Q2 <- as.Q4(c(sqrt(2)/2,0,sqrt(2)/2 ,0))

##add rotations means multiply quaternions
Q3 = Q1 + Q2 #this is a quaternion multiplication!
Q4 =Q2 + Q1
```

```
> mis.angle(Q3); mis.angle(Q4)
[1] 2.094395
[1] 2.094395
> mis.axis(Q3); mis.axis(Q4)
          [,1]      [,2]      [,3]
[1,] 0.5773503 0.5773503 0.5773503
           [,1]      [,2]      [,3]
[1,] -0.5773503 0.5773503 0.5773503
>sqrt(sum(Q4^2))
[1] 1
```

Q3 corresponds to rotation of 120º around axis (1,1,1)
Q4 corresponds to rotation of 120º around axis (-1,1,1)

How far did we get?

Quaternions for 3D rotations are:

- Compact representation
- Simple calculation
- Robust composition
- Robust interpolation ?
  - ➢We need some more algebra

Identity and Inverse

Identity quaternion is $(1, 0, 0, 0)$

- applies no rotation
- remains at reference orientation

$\mathbf{q}^{-1}$ is inverse

$\mathbf{q} \cdot \mathbf{q}^{-1}$ gives identity quaternion

What is $\mathbf{q}^{-1}$?

What is the inverse?

$(w, \mathbf{v})^{-1}$ = ($\cos(\theta/2)$, $\sin(\theta/2) \cdot \mathbf{r}$) $^{-1}$

$$(w, \mathbf{v})^{-1} = (\cos(-\theta/2), \sin(-\theta/2)\mathbf{r})$$

$$(w, \mathbf{v})^{-1} = (\cos(\theta/2), -\sin(\theta/2)\mathbf{r})$$

$$(w, \mathbf{v})^{-1} = (w, -\mathbf{v})$$

Only true if $\mathbf{q}$ is normalized

- i.e. $\mathbf{r}$ is a unit vector

Otherwise scale by

$$1/(x^2 + y^2 + z^2 + w^2)$$

Inverse is same axis but opposite angle!

More formally:

$$q^{-1} = \frac{q^*}{qq^*}$$

If $|q|$ = 1

$$q^{-1} = q^*$$

Be careful!

- do not confuse $-q$ with q^*
- If $|q|$ = 1, q and $-q$ represent the same rotation (or almost)

How to rotate a vector with a quaternion?

Main problem:

$$p \in \mathbb{R}^3$$

But:

$$q \in \mathbb{R}^4$$

How do we deal with this?

Practical formula. Like this:

- Treat **p** as quaternion (0, **p**)
- Rotation of **p** by **q** is p'= $\mathbf{q}\,\mathbf{p}\,\mathbf{q}^{-1}$
- Result in the form (0, **p'**)

How to rotate a vector with a quaternion?

Why does this formula work?

Proof:

https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation#Proof_of_the_quaternion_rotation_identity

Intuition:

- First multiply rotates halfway and into 4th dimension
- Second multiply rotates rest of the way, back into 3rd

How to rotate a vector with a quaternion?

Combine with composition?

Assume: $q = q_2 q_1, p' = qpq^{-1}$

Then:

$$(q_2 q_1) p (q_2 q_1)^{-1}$$

- Given the fact that q has module 1: $q^{-1} = q^*$
- In general: $q_1{}^* q_2{}^* = (q_2 q_1)^*$

The result is:

$$p' = q_2 (q_1 p q_1{}^{-1}) q_2{}^{-1}$$

Conclusion:

- Rotation composition also applies to vectors

Be careful:

- To rotate apply multiplications from right to left

Doubt: Why is the angle half?

Several reasons. Here are 2.

The actual rotation is defined by $x' = q\,x\,q^*$

You get a $\theta/2$ from q on the left, and another $\theta/2$ from q^* on the right, which adds up to a θ

If instead of

$$\cos\frac{\theta}{2} + \sin\frac{\theta}{2}\vec{u}$$

it were

$$\cos\theta + \sin\theta\,\vec{u}$$

then rotation of π about any axis would give you the same result

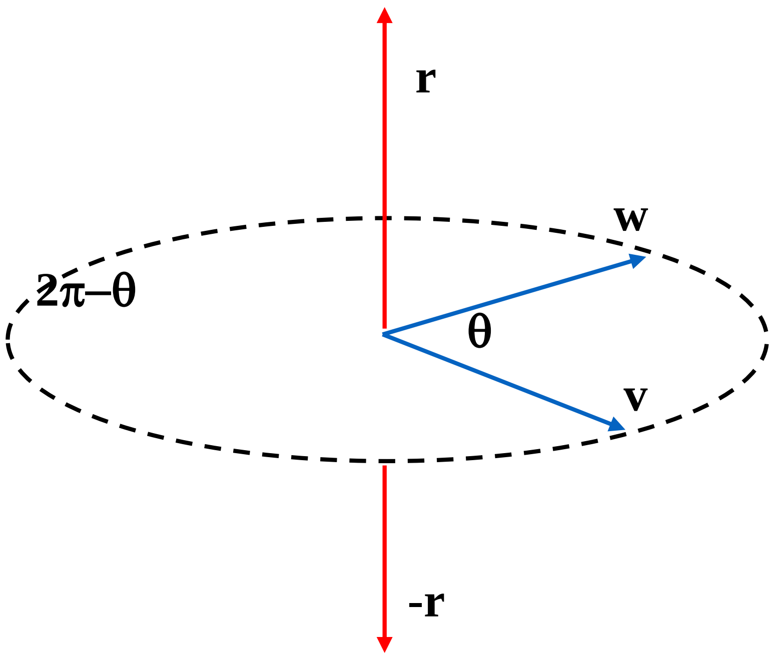
Video tutorial

Useful to review what are quaternions
and how they relate to rotations

https://www.youtube.com/watch?v=d4EgbgTm0Bg

Interpolation

- $\mathbf{q}$ and $-\mathbf{q}$ rotate vector to same place (almost)
- Why almost
  - Not in the same way
  - Important for interpolation



ESCOLA DE NOVES TECNOLOGIES INTERACTIVES
UNIVERSITAT DE BARCELONA

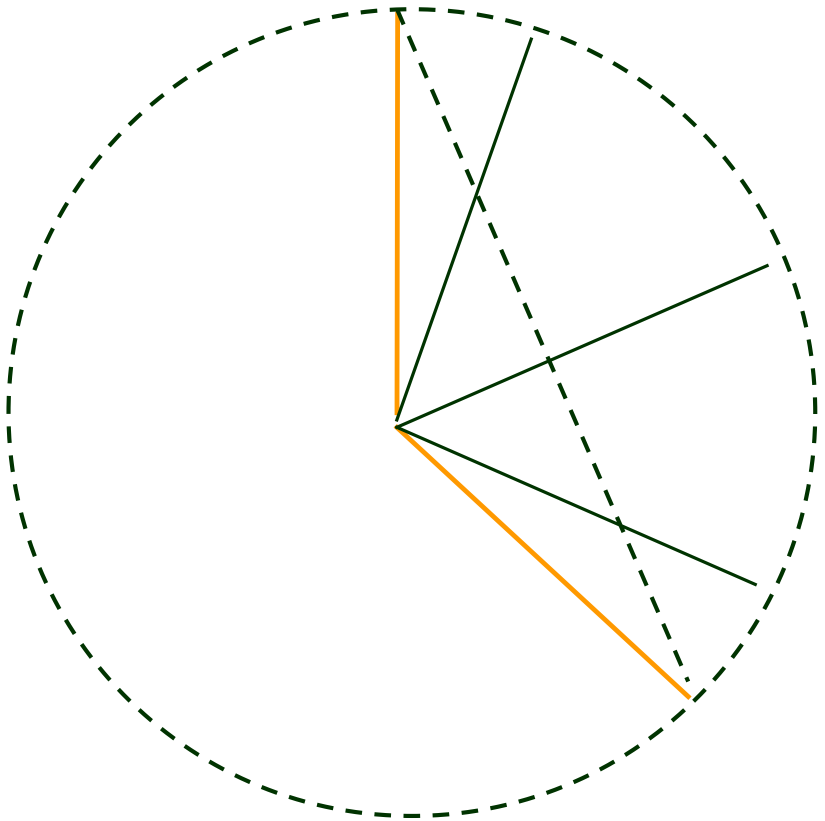
Linear Interpolation?

- Familiar formula

  (1-t) p + t q
- Some trouble
  - Cuts across sphere
  - Moves faster in the middle
  - Resulting quaternions aren't normalized
- Vector3.lerp?

  Will not work
- Vector3.lerp + normalization?

  Will not give uniform movement



Spherical Linear Interpolation (Slerp)

- There *is* a (kind of) nice formula for slerp:

$$\text{slerp}(\mathbf{p},\mathbf{q}:t) = \frac{\sin((1-t)\alpha)}{\sin\alpha}\mathbf{p} + \frac{\sin(t\alpha)}{\sin\alpha}\mathbf{q}$$

If p,q are unit quaternions, and α is the angle between them

But:

- Lots of rounding error
- Potential instabilities (divide by 0)

Spherical Linear Interpolation (faster slerp)

Idea: correct t in Lerp to have speed close to Slerp (From Jon Blow's column, *Game Developer*, March 2002)

- Use simple spline to modify t (adjust speed)
- Near to lerp speed, close to slerp precision

```
float f = 1.0f - 0.7878088f*cosAlpha;
    float k = 0.5069269f;
    f *= f;
    k *= f;
    float b = 2*k;
    float c = -3*k;
    float d = 1 + k;
    t = t*(b*t + c) + d;
```

Spherical Linear Interpolation (Conclusion)

- If small steps (or mocap):

  Lerp + normalize is good enough

- If bigger interpolation:

  Fast Slerp

Be careful!

- If dot product of 2 quaternions is negative ($\cos(\alpha) < 0$), it means:

$$\alpha > 180$$

➢ You are interpolating the long way

➢ You want to take –p instead of p to interpolate the short way

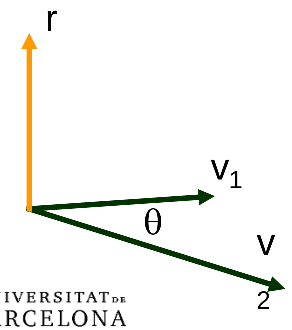
Quaternion as rotations

Have vector $\mathbf{v}_1$, want to rotate to $\mathbf{v}_2$

Need rotation vector $\mathbf{r}$, angle θ

Plug into previous formula

$$\theta = \operatorname{acos}(\hat{\mathbf{v}}_1 \bullet \hat{\mathbf{v}}_2)$$
$$\mathbf{r} = \mathbf{v}_1 \times \mathbf{v}_2$$



Common trick – originally from
Game Gems 1 (Stan Melax)

Use trig identities to avoid acos

- Normalize $\mathbf{v}_1$, $\mathbf{v}_2$

$$\mathbf{r} = \hat{\mathbf{v}}_1 \times \hat{\mathbf{v}}_2 \qquad s = \sqrt{2(1 + \hat{\mathbf{v}}_1 \bullet \hat{\mathbf{v}}_2)}$$

Build quat

$$\mathbf{q} = (2s, \mathbf{r} / s)$$

- More stable when $\mathbf{v}_1$, $\mathbf{v}_2$ near parallel

ESCOLA DE NOVES TECNOLOGIES INTERACTIVES

UNIVERSITAT DE BARCELONA

How far did we get? (revisited)

Quaternions for 3D rotations are:

- Compact representation
- Simple calculation
- Robust composition
- Robust interpolation

Implementation Advice

If you only use quaternions only for rotations:

- Normalize to reduce floating point errors
- Use tricks to avoid functions such as arccos and arctan
- Check collinearity (dot prod = 0) before performing cross prod (infinite)

Many Benefits

- Avoids *Gimbal Lock*
- *Simpler* algorithms to combine successive rotations (compared to using rotation matrices)
- Easier to normalize than rotation matrices
- Interpolation is feasible
- Mathematically stable – suitable for statistics

Conclusions

- Quaternions are good
- Quaternions are nice
- Quaternions are precise

- You should study and master quaternion operations
- You should implement the use of quaternions for rotations

Next weeks

- Direct kinematics (with quaternions)
- Direct Movement interpolation (with quaternions)
- Inverse Kinematics principles (with quaternions)
- IK algo 1 (CCD) (with quaternions)
- IK algo 2 (gradient) (with quaternions)
- Constraints in IK (with quaternions)

Online References Used

Jim Van Hearth, from GDC 2009

https://www.essentialmath.com/tutorial.htm

Mathias Sunardi 2006

http://studylib.net/doc/9456410/quaternions

https://www.docsity.com/en/animation-lecture-slides-computer-graphics-and-animation-2/35739/

Pictures also from wikipedia