

Why are quaternions used for rotations?

Asked 12 years, 11 months ago Modified 9 months ago

Viewed 63k times



141



I'm a physicist, and have been learning some programming, and have come across a lot of people using quaternions for rotations instead of writing things in matrix/vector form.

In physics, there are very good reasons we don't use quaternions (despite the bizarre story that's occasionally told about Hamilton/Gibbs/etc). Physics requires that our descriptions have good analytic behavior (this has a precisely defined meaning, but in some rather technical ways that go far beyond what's taught in normal intro classes, so I won't go into any detail). It turns out that quaternions don't have this nice behavior, and so they aren't useful, and vectors/matrices do, so we use them.

However, restricted to rigid rotations and descriptions that do not use any analytic structures, 3D rotations can be equivalently described either way (or a few other ways).

Generally, we just want a mapping of a point $X = (x, y, z)$ to a new point $X' = (x', y', z')$ subject to the constraint that $X^2 = X'^2$. And there are lots of things that do this.

The naive way is to just draw the triangles this defines and use trigonometry, or use the isomorphism between a point (x, y, z) and a vector (x, y, z) and the function $f(X) = X'$ and a matrix $MX = X'$, or using quaternions, or projecting out components of the old vector along the new one using some other method $(x, y, z)^T \cdot (a, b, c)$ (x', y', z') , etc.

From a math point of view, these descriptions are all equivalent in this setting (as a theorem). They all have the same number of degrees of freedom, the same number of constraints, etc.

So why do quaternions seem to preferred over vectors?

The usual reasons I see are no gimbal lock, or numerical issues.

The no gimbal lock argument seems odd, since this is only a problem of euler angles. It is also only a coordinate problem (just like the singularity at $r=0$ in polar coordinates (the Jacobian loses rank)), which means it is only a local problem, and can be resolved by switching coordinates, rotating out of the degeneracy, or using two overlapping coordinate systems.

I'm less sure about numerical issues, since I don't know in detail how both of these (and any alternatives) would be implemented. I've read that re-normalizing a quaternion is easier than doing that for a rotation matrix, but this is only true for a general matrix; a rotation has additional constraints that trivializes this (which are built

into the definition of quaternions) (In fact, this has to be true since they have the same number of degrees of freedom).

So what is the reason for the use of quaternions over vectors or other alternatives?

matrix

3d

quaternions

rotational-matrices

Share

Improve this question

Follow

edited Jun 10, 2018 at 6:05



Mateen Ulhaq

27.1k ● 21 ● 117 ● 152

asked Jan 18, 2012 at 23:30



JMP

1,547 ● 2 ● 10 ● 5

3 The "no gimbal lock" thing is a lie anyway. You have the same gimbal lock problem that you have with Euler angles if you use two orthogonal rotations with a quaternion. You only don't have an issue for a single rotation since it is 1 operation, not 3. – [Damon](#) Nov 15, 2013 at 16:40

6 @Damon This is not completely true. See mathoverflow.net/a/95908/97344 – [plasmacel](#) Dec 10, 2016 at 17:21

9 Answers

Sorted by:

Highest score (default)





83



Gimbal lock is one reason, although as you say it is only a problem with Euler angles and is easily solvable. Euler angles are still used when memory is a concern as you only need to store 3 numbers.

For quaternions versus a 3x3 rotation matrix, the quaternion has the advantage in size (4 scalars vs. 9) and speed (quaternion multiplication is much faster than 3x3 matrix multiplication).

Note that *all* of these representations of rotations are used in practice. Euler angles use the least memory; matrices use more memory but don't suffer from Gimbal lock and have nice analytical properties; and quaternions strike a nice balance of both, being lightweight, but free from Gimbal lock.

Share Improve this answer

answered Jan 18, 2012 at 23:38

Follow




Peter Alexander


54.2k ● 14 ● 121 ● 169

1 But a rotation matrix doesn't have that many independent components--it's constrained. A two dimensional rotation is specified by three coordinates in three dimensions, regardless of representation. Matrices have more components in general because they can do more than rotations. But in the case of rotations the extra components are determined in terms of the others. – [JMP](#) Jan 18, 2012 at 23:45

3 @JMP: You're right. A lot of people do "compress" the matrix so that you only store as much information as needed, but a compressed matrix is more difficult to deal with, so you lose

out on performance. It's all about trade-offs in memory and performance. – [Peter Alexander](#) Jan 18, 2012 at 23:50

14 @JMP Standard matrix multiplication routines need all 9 values, though. Even though only 3 of them are independent, it still takes 9 numbers' worth of memory when you go to actually do the math (again, if you're actually doing the matrix multiplication in the computer). – [David Z](#) Jan 18, 2012 at 23:56 

6 "quaternion multiplication is much faster than 3x3 matrix multiplication" Really? Quaternion rotation requires 24 add/mul operations (due to twice cross-product and supplemental operations), 3x3 matrix requires only 15 add/mul operations. – [Marat Bukharov](#) Aug 6, 2019 at 20:41 

2 @MaratBuharov To my understanding rotating a vector is more expensive with a quaterion than with a matrix but the quaternion is faster for combining a chain of rotations (something that is quite extensively done, e.g. in robotics when transforming between different links in the kinematic chain). See for example math.stackexchange.com/a/1355206/828555 – [luator](#) Feb 7, 2023 at 9:52



49



In physics, there are very good reasons we don't use quaternions (despite the bizarre story that's occasionally told about Hamilton/Gibbs/etc).

Physics requires that our descriptions have good analytic behavior (this has a precisely defined meaning, but in some rather technical ways that go far beyond what's taught in normal intro classes, so I won't go into any detail). It turns out

that quaternions don't have this nice behavior, and so they aren't useful, and vectors/matrices do, so we use them.

Well, I am a physicist, too. And there are some situations where quaternions simply rock! Spherical Harmonics for example. You have two atoms scattering, exchanging an electron: what is the orbital spin transfer? With quaternions it is just multiplication i.e. summing up the exponents of the SH base functions expressed as quaternions. (Getting the Legendre Polynomials into quaternion notation is a bit tedious though).

But I agree, they are not a universal tool, and especially in rigid body mechanics they would be very cumbersome to use. Yet to cite Bertrand Russell answer in question of a student how much math a physicist needs to know: "*As much as possible!*"

Anyway: Why do we love quaternions in computer graphics? Because they have a number of appealing properties. First one can nicely interpolate them, which is important if one is animating rotating things, like the limbs around a joint. With a quaternion it is just scalar multiplication and normalization. Expressing this with a matrix requires evaluation of sin and cos, then building a rotation matrix. Then multiplying a vector with a quaternion is still cheaper as going through a full vector-matrix multiplication, it is also still cheaper if one adds a translation afterwards. If you consider a skeletal animation system for a human character, where one must

evaluate a lot of translation/rotations for a large number of vertices, this has a huge impact.

Another nice side effect of using quaternions is, that any transformation inherently is orthonormal. With translation matrices one must re-orthonormalize every couple of animation steps, due to numerical round-off errors.

Share Improve this answer

Follow

edited Oct 11, 2014 at 0:53



matthias_h

11.4k ● 9 ● 23 ● 40


answered Jan 19, 2012 at 10:20



datenwolf

162k ● 13 ● 191 ● 310

1 Do you have a reference for spherical harmonics / Legendre polynomials with quaternions? I'm about to submit a paper dealing with related topics and would love to see (be able to cite) other work on this. – [Mike](#) Feb 12, 2013 at 18:07

7 @Mike: Out of my head, unfortunately nothing published. Unfortunately quaternions are still rather obscure to physicists. I just remember it, because my tutor of Quantum Mechanic 2 made this an exercise and I was blown away by it. What we essentially did was using the term $\exp((a \cdot i\omega + b \cdot j\theta + c \cdot k\eta + d)r)$, where r itself was a complex variable. If you plot this you get a 3 dimensional distribution (we had to develop the exponential series with respect to a quaternion variable first). This allows for doing a "fourier" transform, resulting in something you could turn into the known SH terms. – [datenwolf](#) Feb 12, 2013 at 18:38 



42



The no gimbal lock argument seems odd, since this is only a problem of euler angles. It is also only a coordinate problem (just like the singularity at $r=0$ in polar coordinates (the Jacobian loses rank)), which means it is only a local problem, and can be resolved by switching coordinates, rotating out of the degeneracy, or using two overlapping coordinate systems.

Many 3D applications like using Euler angles for defining an object's orientation. For flight-sims in particular, they represent a theoretically useful way of storing the orientation in a way that is easily modifiable.

You should also be aware that things like "switching coordinates, rotating out of the degeneracy, or using two overlapping coordinate systems" all require effort. Effort means code. And code means performance. Losing performance when you don't *have* to is not a good thing for many 3D applications. After all, what is to be gained by all of these tricks, if just using quaternions would get you everything you needed.

I'm less sure about numerical issues, since I don't know in detail how both of these (and any alternatives) would be implemented. I've read that re-normalizing a quaternion is easier than doing that for a rotation matrix, but this is only true for a general matrix; a rotation has additional

constraints that trivializes this (which are built into the definition of quaternions) (In fact, this has to be true since they have the same number of degrees of freedom).

The numerical issues come up when dealing with multiple consecutive rotations of an orientation. Imagine you have an object in space. And every timeslice, you apply a small change of yaw to it. After each change, you need to re-normalize the orientation; otherwise, precision problems will creep in and screw things up.

If you use matrices, each time you do matrix multiplication, you must re-orthonormalize the matrix. The matrix that you are orthonormalizing is not *yet* a rotation matrix, so I wouldn't be too sure about that easy orthonormalization. However, I can be sure about this:

It won't be as fast as a 4D vector normalization. That's what quaternions use to normalize after successive rotations.

Quaternion normalization is cheap. Even specialized rotation matrix normalization will not be *as* cheap. Again, performance matters.

There's also another issue that matrices don't do easily: interpolation between two different orientations.

When dealing with a 3D character, you often have a series of transformations defining the location of each

bone in the character. This hierarchy of bones represents the character in a particular pose.

In most animation systems, to compute the pose for a character at a particular time, one interpolates between transformations. This requires interpolating the corresponding transformations.

Interpolating two matrices is... non-trivial. At least, it is if you want something that resembles a rotation matrix at the end. After all, the purpose of the interpolation is to produce something part-way between the two transformations.

For quaternions, all you need is a 4D lerp followed by a normalize. That's all: take two quaternions and linearly interpolate the components. Normalize the result.

If you want better quality interpolation (and sometimes you do), you can bring out the [spherical lerp](#). This makes the interpolation behave better for more disparate orientations. This math is *much* more difficult and requires more operations for matrices than quaternions.

Share Improve this answer

answered Jan 19, 2012 at 0:24

Follow



Nicol Bolas

472k ● 64 ● 831 ● 1k

I remember the quaternion part of the flight simulator code. I asked how it worked, having never seen them before, and after a long pause was told to not mess with that bit of code! I have used them since and now understand what the pause was about, impure quaternions and two operations to get a

meaningful rotation with complement quaternions involved.

Still, wish he had had the time to explain it to me though!

– [user50619](#) Mar 9, 2021 at 14:48 

$R = R + (R - R * R' * R)/2$ will re-orthogonalize a rotation matrix. This quadratically convergent formula requires **NO square roots**, so it is quite inexpensive. One (or two) iterations will fix up any accumulated round-off errors. It is not really the difficult task that is so often asserted by quaternion enthusiasts. – [greg](#) Dec 27, 2023 at 15:50

@greg: "*This quadratically convergent formula requires NO square roots, so it is quite inexpensive.*" Reciprocal square roots are way faster these days than two matrix additions, 2 matrix multiplies, and a matrix division by a scalar.

– [Nicol Bolas](#) Dec 27, 2023 at 16:10 



Opinion: Quaternions are nice.

14



Rotation matrix: *Minor disadvantage:* Multiplication of matrices is ~2 times slower than quaternions. *Minor*

Advantage: Matrix-vector multiplication is ~2 times faster, and large. **Huge disadvantage:** Normalization! Ghram-



Shmit is asymmetrical, which does not give a higher order accurate answer when doing differential equations. More sophisticated methods are very complex and expensive.



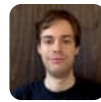
Axis (angle = length of axis) *Minor advantage:* Small.

Moderate disadvantage: Multiplication and applying to a vector is slow with trig. *Moderate disadvantage:* North-pole singularity at length = 2π , since all axis directions do nothing. More code (and debugging) to automatically rescale it when it gets near 2π .

Share Improve this answer

answered Apr 26, 2013 at 23:47

Follow



Kevin Kostlan

3,509 ● 7 ● 30 ● 36



10

The usual reasons I see are no gimble lock, or numerical issues.



And they are good reasons.



As you already seem to understand, quaternions encode a single rotation around an arbitrary axis as opposed to three sequential rotations in Euler 3-space. This makes quaternions [immune to gimbal lock](#).

Also, some forms of interpolation become nice and easy to do, like [SLERP](#).

...or using two overlapping coordinate systems.

From a performance perspective, why is your solution better?

I could go on, but quaternions are just one possible tool to use. If they do not suit your needs, then do not use them.

Share Improve this answer

edited Aug 27, 2016 at 4:24

Follow



Yet rotation matrices do the same, as well as have more algebraic properties that can be neatly used. On top of that matrix manipulation is one of the things computers are particularly good at. – [paul23](#) Dec 24, 2018 at 5:23



10



Generally, we just want a mapping of a point $X = (x, y, z)$ to a new point $X' = (x', y', z')$ subject to the constraint that $X^2 = X'^2$. And there are lots of things that do this.



We absolutely ***do not just*** want that. There is a very important subtlety that lots of people miss (see, e.g., [this reference](#) that sounds authoritative, but totally misses the point). The construction you're talking about (draw the triangles and use trig, etc.) will correctly rotate *one* vector into *one* other vector. But there are infinitely many rotations that will do this. In particular, I can come along after you've done your rotation, and then rotate the whole system around the X' vector by any amount. That won't change the position of X' at all, but will change the position of any vector that isn't just a multiple of X' . The combination of your rotation and mine is equivalent to another single rotation (since rotations [form a group](#)). In general, you need to be able to represent any such rotation.

It turns out that you *could* do this with just a vector — but not in the way that you're talking about. That's the [axis-angle representation of rotations](#). And combining rotations in the axis-angle representation is difficult. Quaternions make it easy, along with lots of other things. Basically, quaternions have all the advantages of other representations, and none of the drawbacks. (Though I'll admit that there may be specific applications for which some other representation may be better.)

Another point that I think a lot of people miss is that quaternions are often better when you need to do operations *on the rotations themselves*. Even to get to the point where we might apply a rotation to a vector, we need to figure out what that rotation should be. Very often we will need to **compose** rotations, **invert** rotations, and **interpolate** or **extend** rotations. All of these can be done very efficiently and accurately with quaternions. Other representations may be almost as good as quaternions for one of these operations, but certainly not all of them.

Share Improve this answer

edited Feb 5, 2023 at 2:02

Follow

answered Feb 14, 2013 at 1:45



Mike

20.2k ● 13 ● 62 ● 95



8



It's worth bearing in mind that all the properties related to rotation are not truly properties of Quaternions: they're properties of *Euler-Rodrigues Parameterisations*, which is the actual 4-element structure used to describe a 3D rotation.

Their relationship to Quaternions is purely due to a paper by Cayley, "On certain results related to Quaternions", where the author observes the correlation between Quaternion multiplication and combination of Euler-Rodrigues parameterisations. This enabled aspects of Quaternion theory to be applied to the representation of rotations and especially to interpolating between them.

You can read the paper here:

<https://archive.org/details/collmathpapers01caylrich> . But at the time, there was no connection between Quaternions and rotation and Cayley was rather surprised to find there was:

In fact the formulae are precisely those given for such a transformation by M. Olinde Rodrigues Liouville, t. v., "Des lois géométriques qui régissent les déplacements d'un système solide [...]" (or Comb. Math. Journal, t. iii. p. 224 [6]). It would be an interesting question to account, a priori, for the appearance of these coefficients here.

However, there is nothing intrinsic about Quaternions that gives any benefit to rotation. Quaternions do not avoid gimbal lock; Euler-Rodrigues parameterisations do. Very few computer programs that perform rotation are likely to truly implement Quaternion types that are first-class complex mathematical values. Unfortunately, a misunderstanding of the role of Quaternions seems to have leaked out somewhere resulting in quite a few baffled graphics students learning the details of complex math with multiple imaginary constants and then being baffled as to why this solves the problems with rotation.

Share Improve this answer

Follow

edited Jun 8, 2016 at 23:33



baptiste

77k ● 21 ● 204 ● 299

answered Jul 13, 2015 at 19:08



Mark Green

1,330 ● 12 ● 19



5



An answer that someone might read: There are tedious problems with all representations. Quaternions are smaller than matrices but the quaternion multiplication is not a mere vector dot product or such, and in fact takes more time on a computer than the dot product of two 3x3 matrices. (Computers are very very good at operating with ordinary matrices)

Matrices though have other annoying features. For example, they are not stable creatures in the long run. When modelling rotations in 3D space, one usually

accumulates rotations on top of each other into an Orientation matrix, that is just a single rotation matrix storing the orientation of a reference frame. This process will over the course of millions of additions cause the O-matrix to diverge from a strict rotation matrix form. This can be circumvented by periodically reconfiguring the matrix, but there are conditions when this is nontrivial. Namely the no-rotation case of identity matrix.

You would want to find an axis-angle representation (or quaternion representation) of the rotation, and then reproduce a matrix for that. Most algorithms produce a zero vector, and then encounter zero-division in this case. In these kinds of cases it is also generally a poor idea to try to avoid such cases with "if 0 then..." -type of solutions, since a) forks are slow and b) you can still end up machine epsilon apart from singularity and end up with horrendous errors.

Share Improve this answer

answered Nov 21, 2019 at 7:30

Follow



Elmore

276 ● 4 ● 7



1



Unit quaternions provide a compact representation of the Orthogonal Group of three dimensions, $O(3)$, and especially its subgroup, the Special Orthogonal Group of three dimensions $SO(3)$. This has many uses, but the one I know the best is its uses in Inertial Navigation systems using a Strapdown arrangement of accelerators and gyros. An element of $SO(3)$ is used to represent a



vehicles 'position' on the Earth (or rather a sphere that approximates the surface of the earth.) Another is used to specify the vehicle's 'attitude,' which is to say the relation of its bodyframe with the local tangent plane. Both of these are 'integrated' with small changes (10Hz or smaller time increments) to update them with new data. The 'attitude' and 'position' rotations form a "coupled" differential systems, since forces applied to the body of the vehicle are resolved in the local tangent plane to update the vehicle 'position' rotation.

Quaternions are easily integrated in this fashion (four quaternion additions.) After "integration" the result will no longer be a member of $SO(3)$ but a simple re-normalization "projects" the integrated quaternion onto $SO(3)$ again. One only uses and keeps 4 values per quaternion.

The utilization of quaternion integration to effect inertial navigation was already an industry standard in the 1980s. The first paper of which I am aware proposing this was published in 1973. At the time mathematics students were only introduced to quaternions as an "odd" example in algebra (a non-commutative division ring.)

However, quaternions are now the preferred representation of $SO(3)$ in many applications (e.g., computer graphics.) From a mathematical perspective it is even more interesting as it provides possible the simplest example of 'calculus on manifolds!' (the sphere is a simple manifold and $SO(3)$ is a Lie Group.)

Share Improve this answer

answered Jun 6, 2021 at 10:15

Follow



PHILLIP CHESSON

11 ● 1
