Quaternions for 3D Rotations: A plain, step-by-step treatment (examples before formulas)

Abstract

We present a short, practical note on representing and composing 3D rotations for simulation and control. We start with the problem statement, show the classic Euler angle and matrix approaches and their problems, and then give a clear, elementary development of quaternion rotations. We explain how quaternions extend complex numbers, how to rotate vectors, how to convert between local and global axes, and how to convert any rotation from one basis to any other basis. We include explicit algebraic derivations and a worked numeric example (rotate about global X by 90°, then about the new local Y by 90°).

1 Define the problem (plain words)

Suppose we have an object — a robot arm, a drone, a printer pen, or an insect wing. We need to:

- represent the object's orientation (its *state*),
- apply rotations about global (world) axes or about the object's current local axes,
- compose many rotations, interpolate smoothly, and avoid singularities or numerical drift.

This note shows a representation that makes these tasks easy and efficient.

2 Traditional methods (short)

2.1 Euler angles

Euler angles describe orientation as three sequential rotations about coordinate axes. Example: intrinsic (local) convention $X\,Y'\,Z''$:

1. Rotate about X by α. Axes become X', Y', Z'.
2. Rotate about the new Y' by β. Axes become X'', Y'', Z''.

3. Rotate about the new Z'' by γ.

This is clear to a human, but:

- order matters (non-commutative),
- there can be gimbal lock,
- interpolating angles does not produce smooth physical motion.

2.2 Rotation matrices

Rotation matrices are unambiguous but redundant (9 numbers, 3 DOF) and can lose orthogonality in numerical integration. They do not solve the intuition problem about "local vs global" rotations as cleanly as we would like.

So: we now build up quaternions carefully, starting from complex numbers.

3 A short refresher: imaginary numbers and Euler's formula

An imaginary unit i satisfies $i^2 = -1$. Complex numbers $z = a + bi$ can be used to rotate the plane. Euler's formula for complex numbers:

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

Multiplying a complex number by $e^{i\theta}$ rotates it by angle θ in the plane.

4 Quaternions as an extension of complex rotations

4.1 Pure/quaternionic "imaginary" units

A quaternion has a scalar part and a vector part:

$$q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k} \equiv (w, \mathbf{v}), \quad \mathbf{v} = (x, y, z).$$

A *pure* quaternion is one with $w = 0$; its vector part behaves similarly to an imaginary direction. For any unit vector $\hat{u}$ (viewed as a pure quaternion), we have

$$\hat{u}^2 = -1.$$

This is the quaternion analogue of $i^2 = -1$.

We use the standard multiplication rules:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1,$$

and the anti-commutation rules:

$$\mathbf{ij} = \mathbf{k}, \quad \mathbf{ji} = -\mathbf{k}, \quad \mathbf{jk} = \mathbf{i}, \quad \mathbf{kj} = -\mathbf{i}, \quad \mathbf{ki} = \mathbf{j}, \quad \mathbf{ik} = -\mathbf{j}.$$

These identities are used in the explicit FOIL computations below.

4.2 Quaternion exponential: the direct analogue

For any unit pure quaternion $\hat{u}$ and any real θ,

$$\boxed{e^{\hat{u}\theta} \;=\; \cos\theta \;+\; \hat{u}\,\sin\theta}$$

This is the direct analogue of the complex Euler formula. (We have not yet said how this relates to rotating a 3D vector — we build to that.)

Important note: Right now think of $\hat{u}$ as a fixed imaginary direction in quaternion space; we purposely do not call $\hat{u}$ an "axis" in the 3D sense yet — we are building intuition.

5 Using $e^{\hat{u}\theta}$ to rotate — the key idea

If you multiply a quaternion by $e^{\hat{u}\theta}$, you get another quaternion. But when you try to rotate a 3D vector represented as a pure quaternion by simply multiplying on one side, the result is not a pure quaternion (it acquires a scalar part). To get a rotated *vector* in pure quaternion form you must *sandwich* the vector between a quaternion and its inverse.

We now reveal the core identity you will use (and why the half-angle appears):

If $q = e^{\hat{u}\alpha} = \cos\alpha + \hat{u}\sin\alpha$, then for any pure quaternion $v = (0, \mathbf{v})$,

$$q\,v\,q^{-1} = \big(0,\, \mathbf{v}\cos(2\alpha) + (\hat{u}\times\mathbf{v})\sin(2\alpha) + \hat{u}(\hat{u}\cdot\mathbf{v})\big(1-\cos(2\alpha)\big)\big).$$

That right-hand side is exactly the Rodrigues rotation formula for rotating $\mathbf{v}$ by angle 2α about direction $\hat{u}$. Thus:

$$\boxed{\text{The sandwich } qvq^{-1} \text{ with } q = e^{\hat{u}\alpha} \text{ rotates } \mathbf{v} \text{ by } 2\alpha \text{ about } \hat{u}.}$$

Therefore, to rotate by an angle θ about $\hat{u}$, pick $\alpha = \frac{\theta}{2}$ and set

$$\boxed{q(\hat{u},\theta) \;=\; e^{\hat{u}\,\theta/2} \;=\; \cos\frac{\theta}{2} \;+\; \hat{u}\,\sin\frac{\theta}{2}}.$$

Then $q(\hat{u},\theta)\, v\, q(\hat{u},\theta)^{-1}$ rotates $\mathbf{v}$ by θ about $\hat{u}$. We built up to the half-angle: it follows from the identity above that the sandwich with exponent α rotates by 2α.

6 Worked example: rotate about x by 90° — FOIL with $\mathbf{i}, \mathbf{j}, \mathbf{k}$

This is the important example. We will FOIL everything explicitly using $\mathbf{i}, \mathbf{j}, \mathbf{k}$ multiplications so there is no hand-waving.

6.1 Goal

Rotate the vector $\mathbf{v} = (0, 1, 0)$ by $\theta = \frac{\pi}{2}$ ($90°$) about the x-axis.

6.2 Build the quaternion

Half-angle $\theta/2 = \pi/4$. Set

$$a = \cos\tfrac{\pi}{4} = \tfrac{\sqrt{2}}{2}, \qquad b = \sin\tfrac{\pi}{4} = \tfrac{\sqrt{2}}{2}.$$

Quaternion for half-angle about x (pure unit $\hat{i}$) is

$$q_x = a + b\mathbf{i}.$$

Vector as pure quaternion:

$$v = 0 + 0\mathbf{i} + 1\mathbf{j} + 0\mathbf{k} = \mathbf{j}.$$

6.3 Compute $q_x v$

$$q_x v = (a + b\mathbf{i})\mathbf{j} = a\mathbf{j} + b(\mathbf{ij}).$$

Use $\mathbf{ij} = \mathbf{k}$, so

$$q_x v = a\mathbf{j} + b\mathbf{k}.$$

6.4 Compute q_x^{-1} and multiply

Because q_x is unit, $q_x^{-1} = q_x^* = a - b\mathbf{i}$.

Now compute

$$q_x v q_x^{-1} = (a\mathbf{j} + b\mathbf{k})(a - b\mathbf{i}).$$

Expand (FOIL):

$$(a\mathbf{j} + b\mathbf{k})(a - b\mathbf{i}) = a^2\mathbf{j} - ab\,\mathbf{ji} + ab\,\mathbf{k} - b^2\mathbf{ki}.$$

Use multiplication rules:

$$\mathbf{ji} = -\mathbf{k}, \qquad \mathbf{ki} = \mathbf{j}.$$

Substitute:

$$\begin{aligned} &= a^2\mathbf{j} - ab(-\mathbf{k}) + ab\mathbf{k} - b^2\mathbf{j} \\ &= (a^2 - b^2)\mathbf{j} + (ab + ab)\mathbf{k} \\ &= (a^2 - b^2)\mathbf{j} + 2ab\,\mathbf{k}. \end{aligned}$$

Now with $a = b = \frac{\sqrt{2}}{2}$:

$$a^2 - b^2 = 0, \qquad 2ab = 1,$$

so

$$q_x v q_x^{-1} = \mathbf{k},$$

i.e. the pure quaternion $(0, 0, 0, 1)$, which corresponds to the vector $(0, 0, 1)$.

This completes the FOILed proof: rotating $(0, 1, 0)$ by $90°$ about x gives $(0, 0, 1)$.

For completeness, we also show the explicit arithmetic with component notation. The quaternion $q_x = (a, b, 0, 0)$, $v = (0, 0, 1, 0)$.

Compute $q_x v$:

$$\begin{aligned} \text{scalar} &= a \cdot 0 - b \cdot 0 - 0 \cdot 1 - 0 \cdot 0 = 0, \\ \text{x} &= a \cdot 0 + b \cdot 0 + 0 \cdot 0 - 0 \cdot 1 = 0, \\ \text{y} &= a \cdot 1 - b \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = a, \\ \text{z} &= a \cdot 0 + b \cdot 1 - 0 \cdot 0 + 0 \cdot 0 = b. \end{aligned}$$

So $q_x v = (0, 0, a, b)$.

Then $q_x^{-1} = (a, -b, 0, 0)$, and $(0, 0, a, b) \cdot (a, -b, 0, 0) = (0, 0, 0, 1)$, as before.

7 State quaternion and mapping local points to global

Start with the global standard state $q = \big(1, 0, 0, 0\big)$ (identity). If we rotate the system about x by $90°$, the state quaternion becomes $q_{\text{state}} = q_x$ from the example above.

Given any point expressed in the object's local coordinates (for example the local vector $(0, 1, 0)$), we can map it to world coordinates by the sandwich:

$$(0, \mathbf{v}_{\text{world}}) \;=\; q_{\text{state}}\,(0, \mathbf{v}_{\text{local}})\,q_{\text{state}}^{-1}.$$

So in our example $q_{\text{state}} = q_x$ maps $(0, 1, 0)_{\text{local}}$ to $(0, 0, 1)_{\text{world}}$ exactly as shown above.

8 Now: rotate about the new local y axis by $90°$

We want to find the new q_{state} after we rotate the object first about global x by $90°$, and then about the object's new local y by $90°$.

8.1 Build up to the solution

It essentially comes down to saying, we know how to rotate about any global axis. Upon the rotation about the x axis, if we can find what global axis corresponds to the local y axis, rotating about that global axis would be equivalent to rotating about the local y axis. Then, the problem comes down to finding what axis corresponds to the local Y axis. But the Y axis is still just a vector, that we can transform from local to global coordinates. And so sandwich multiply the y axis vector with the $q_{\text{state current}}$, get the global axis corresponding to the local Y axis. And define a q_{rot} to rotate about that new axis. Then, like complex numbers, having two successive rotations is equivalent to left multiplying the quaternions.

To see why, suppose you rotate v_0 by q_1 then by q_2. Then

$$v_2 = q_2(q_1 v_0 q_1^{-1})q_2^{-1} = (q_2 q_1)v_0(q_2 q_1)^{-1},$$

which is the quaternionic analogue of composing rotations. Therefore, $q_{\text{state new}} = q_{\text{rot}} \cdot q_{\text{state curr}}$.

Repeatedly reminding the reader the purpose and intent is very useful: this approach allows us to always apply rotations in the global frame, which simplifies generalization to arbitrary basis changes.

8.2 Now do it concretely (example continuation)

We already have $q_{\text{state curr}} = q_x = a + b\mathbf{i}$. The local y axis is the pure quaternion $\mathbf{j}$. Map it to world:

$$\hat{y}_{\text{world}} = \text{Im}(q_x \mathbf{j} q_x^{-1}).$$

From the FOIL above we found $q_x \mathbf{j} q_x^{-1} = \mathbf{k}$, so

$$\hat{y}_{\text{world}} = \mathbf{k} = (0, 0, 1).$$

In essence, what that means to us is that we wanted to rotate around the local y-axis. But since we saw before that the local y-axis corresponds to the global z-axis, rotation around the local y-axis is equivalent to rotating the entire space around the global z-axis, by the required angle θ.
Define the rotation quaternion that rotates about $\mathbf{k}$ by $\theta = \pi/2$:

$$q_{\text{rot}} = q(\mathbf{k}, \tfrac{\pi}{2}) = \cos\tfrac{\pi}{4} + \mathbf{k}\sin\tfrac{\pi}{4} = a + b\mathbf{k}.$$

Then left-multiplying:

$$q_{\text{state new}} = q_{\text{rot}}\, q_{\text{state curr}}.$$

We compute (plugging $a = b = \frac{\sqrt{2}}{2}$):

$$q_{\text{state new}} = (a + b\mathbf{k})(a + b\mathbf{i}) = (a^2) + ab(\mathbf{i} + \mathbf{k}) + b^2(\mathbf{ki}).$$

Using $\mathbf{ki} = \mathbf{j}$ and simplifying gives

$$q_{\text{state new}} = \left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right).$$

8.3 As a side note: an equivalent, elegant solution to simply right multiply by a naive rotation quaternion q_{local}

As a side note, there actually happens to be an equivalent, elegant solution to simply right multiply by a naive rotation quaternion $q_{\text{local}} = a + b\mathbf{j}$:

$$q_{\text{state new}} = q_{\text{state curr}}\, q_{\text{local}}.$$

This yields the same result. However, later we will see that left multiplication is a more generalizable approach for usual coordinate and basis change.

8.4 For the final piece of the act, find what the NEW LOCAL (0,0,1) coordinate corresponds to in the global space, after both these rotations

For the final piece of the act, find what the NEW LOCAL (0,0,1) coordinate corresponds to in the global space, after both these rotations. The new local z-vector is $(0, 0, 0, 1)$. Apply the sandwich with q_{new}:

$$q_{\text{new}} \cdot (0, 0, 0, 1) \cdot q_{\text{new}}^{-1}.$$

Computing this explicitly yields the pure quaternion corresponding to the vector $(1, 0, 0)$ — the global x-axis.

9 Short proof: successive rotations combine by multiplication

Suppose first you rotate vector v_0 by q_1 to obtain v_1:

$$v_1 = q_1 v_0 q_1^{-1}.$$

Then rotate v_1 by q_2 to get v_2:

$$v_2 = q_2 v_1 q_2^{-1} = q_2(q_1 v_0 q_1^{-1})q_2^{-1} = (q_2 q_1)\, v_0\, (q_2 q_1)^{-1}.$$

So applying q_1 then q_2 to v_0 is equivalent to applying the single quaternion $q_2 q_1$. Therefore successive rotations compose by quaternion multiplication — left multiplication when you apply an additional rotation after what's already included on the right.

10 Euler angles implemented with quaternions (left-multiplication approach)

10.1 Global (extrinsic) XYZ using left multiplication

For the extrinsic/global sequence — rotate about global X by α, then about global Y by β, then about global Z by γ — the quaternion product is obtained by left-multiplying in the order of application:

$$q_{\text{total}} = q_z(\gamma)\, q_y(\beta)\, q_x(\alpha).$$

Each q is the quaternion for that angle about that global axis.

Let $q_x(\alpha) = \cos\frac{\alpha}{2} + \mathbf{i}\sin\frac{\alpha}{2}$, $q_y(\beta) = \cos\frac{\beta}{2} + \mathbf{j}\sin\frac{\beta}{2}$, $q_z(\gamma) = \cos\frac{\gamma}{2} + \mathbf{k}\sin\frac{\gamma}{2}$. Then

$$q_{\text{total}} = \big(\cos\tfrac{\gamma}{2} + \mathbf{k}\sin\tfrac{\gamma}{2}\big)\big(\cos\tfrac{\beta}{2} + \mathbf{j}\sin\tfrac{\beta}{2}\big)\big(\cos\tfrac{\alpha}{2} + \mathbf{i}\sin\tfrac{\alpha}{2}\big).$$

The product can be expanded using the multiplication rules to yield the full quaternion components, following the FOIL procedure shown earlier.

10.2 Intrinsic (local) XY'Z" using left multiplication with mapped axes

For the intrinsic sequence, we use the left-multiply approach by mapping local axes to global at each step.

Start with

$$q_x = q(\hat{x}, \alpha)$$

(initial local x = global x, so direct), and set current state $q_1 = q_x$.

Then for the second rotation about local y' (after the first rotation): map the current local y to world using the current state q_1, i.e.,

$$\hat{y}_{\text{world}} = \text{Im}(q_1(0,0,1,0)q_1^{-1})$$

, build $q_{y'} = q(\hat{y}_{\text{world}}, \beta)$, then update $q_2 = q_{y'}\, q_1$.

Then for the third rotation about local z" (after the second): map the current local z to world using the current state q_2, i.e.,

$$\hat{z}_{\text{world}} = \text{Im}(q_2(0,0,0,1)q_2^{-1})$$

, build $q_{z''} = q(\hat{z}_{\text{world}}, \gamma)$, then

$$q_{\text{total}} = q_{z''}\, q_2 = \boxed{q_{z''}\, q_{y'}\, q_x}$$

.

This is our approach using only left-multiplication, which is generalizable.

As a side note, an equivalent elegant solution that works only when converting to global coordinates, and is not generalizable, is to right-multiply by the naive local quaternions in sequence: $q_x\, q_y\, q_z$.

It happens to actually be equivalent, in this case, to the left-multiplication with mapped axes.

11 Generalization: any local axis rotation as a global rotation

Summary of the trick:

1. To rotate about a local axis $\hat{a}_{\text{local}}$ by θ, compute $\hat{a}_{\text{world}} = \text{Im}(q_{\text{state}}(0, \hat{a}_{\text{local}})q_{\text{state}}^{-1})$.
2. Build $g = q(\hat{a}_{\text{world}}, \theta)$.
3. Then applying the local rotation is equivalent to $q_{\text{new}} = g\, q_{\text{state}}$.

This allows you to always express rotations as global left-multiplications if that simplifies your code.

12 General coordinate transform algorithm and change-of-basis quaternion

We didn't have to only change from local to global. The methods above can be generalized to transformations between any two frames.

Given frames A and B with orientations q_{wA} and q_{wB} (map A or B into world), the quaternion that takes vector coordinates from A to B is

$$q_{BA} = q_{wB}^{-1} q_{wA},$$

and for any pure v_A

$$v_B = q_{BA}\, v_A\, q_{BA}^{-1}.$$

For rotation quaternions,

$$r_B = q_{BA}\, r_A\, q_{BA}^{-1}.$$

This is the general change-of-basis approach.

13 Why this step-by-step approach is good (plain bullets)

- Each step uses simple algebra and a single concept (complex rotations → quaternion exponential → sandwich).
- Examples come first, so formulas are discovered and motivated by computation.

- Local vs global is no longer fuzzy: map the axis and left-multiply for a global implementation that generalizes well.
- Conjugation (sandwich) handles both vectors and rotation quaternions uniformly — one algebraic operation to remember.
- Implementation-friendly: quaternions are compact (4 numbers), invertible by conjugate, cheap to multiply, and numerically stable if normalized regularly.
- Quaternions avoid gimbal lock for general rotations, enable smooth interpolation (e.g., SLERP), and are efficient for composition without numerical drift.