A brief summary about quaternion use for describing rotations in 3D space

J. Rabault

30th June 2017

1 Introduction

Quaternions are an appealing tool for describing rotations in 3D as they do not suffer from gimbal lock, that impairs methods based on Euler angle. From a purely 'utilitarian' point of view with the aim of describing rotations in 3D, one just needs to know a few formula about quaternions to apply them.

2 Quaternion formula for description of 3D rotations

- **Quaternion definition**: a quaternion is a collection of 4 real numbers. In all the following, we will use the notation: $q = [q_0, q_1, q_2, q_3]$.
- **Quaternion norm**: a unit quaternion is a quaternion of norm 1, where l the norm of a quaternion is computed as: $l = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$
- **Quaternion scaling**: with a a real number, $a \cdot q = [aq_0, aq_1, aq_2, aq_3]$
- **Sum of two quaternions**: $p + q = [p_0 + q_0, p_1 + q_1, p_2 + q_2, p_3 + q_3]$.
- **Quaternion conjugation**: $q^* = [q_0, -q_1, -q_2, -q_3]$
- **Quaternion inverse**: $q^{-1} = \frac{[q_0, -q_1, -q_2, -q_3]}{q_0^2 + q_1^2 + q_2^2 + q_3^2} = q^*/l^2$. If q is a unit quaternion, $q^{-1} = q^*$.
- **Product of two quaternions**: note that this is not commutative, i.e. in general $p \otimes q \neq q \otimes p$:

$$r = p \otimes q = \begin{bmatrix} p_0 q_0 - p_1 q_1 - p_2 q_2 - p_3 q_3 \\ p_0 q_1 + p_1 q_0 + p_2 q_3 - p_3 q_2 \\ p_0 q_2 - p_1 q_3 + p_2 q_0 + p_3 q_1 \\ p_0 q_3 + p_1 q_2 - p_2 q_1 + p_3 q_0 \end{bmatrix}$$

  where, of course, the first line of the result is r_0, the second one r_1, etc...

- **Conjugate of product**: $(p \otimes q)^* = q^* \otimes p^*$.

- **Description of a rotation in 3D as a quaternion**: all rotations in 3D can be described as unit quaternions, and vice-versa. If R is the rotation in 3D around axis described by the vector $\omega = (\omega_x, \omega_y, \omega_z)$ with $l_\omega = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2}$ its norm, and of angle θ, then it is described by the unit quaternion:

$$q_R = \left[cos(\frac{\theta}{2}), \frac{\omega_x}{l_\omega} sin(\frac{\theta}{2}), \frac{\omega_y}{l_\omega} sin(\frac{\theta}{2}), \frac{\omega_z}{l_\omega} sin(\frac{\theta}{2}) \right]$$

  When using this formula, be careful not to divide by zero (l_{omega} must be different from zero); if the rotation you describe is the identity, the associated quaternion is $[1, 0, 0, 0]$. This formula is also very easy to inverse: you can obtain θ from the first term, and once θ is known computing ω is easy. Note that you can describe a rotation other than identity in two ways as $R(\theta, \omega) = R(-\theta, -\omega)$.

  Note also that the formula for quaternion inverse and conjugation make sense: $R^{-1}(\theta, \omega) = R(\theta, -\omega)$, which corresponds in the case of unit quaternion describing rotations to $q^{-1} = q^*$.

- **Expression of a vector as a quaternion**: if $v = (v_x, v_y, v_z)$ is a vector in 3D, you can write it as a quaternion: $q_v = [0, v_x, v_y, v_z] = [0, \vec{v}]$.

- **Relation between vector quaternions, scalar product and cross product**:

$$[0, \vec{u}] \otimes [0, \vec{v}] = (-\vec{u} \cdot \vec{v}, \vec{u} \times \vec{v})$$

- **Applying a unit quaternion (rotation) on a vector**: if q_R is a unit quaternion, describing the 3D rotation R, and v a vector, then:

$$[0, R(v)] = q \otimes [0, \vec{v}] \otimes q^*$$

- **Relation between quaternion product and matrix composition**: unit quaternion product is the equivalent of rotation composition: if R_1 is described by q_{R_1} and R_2 by q_{R_2}, $R_2 \circ R_1$ is described by $q_{R_2} \otimes q_{R_1}$.

3 A few complementary notes

There are many modules in all programming languages that give access to quaternion objects and computations. For a very simple implementation in Python, together with an example application, see `https://github.com/jerabaul29/IntegrateGyroData`.

Quaternions are very convenient to use but not really efficient for calculating the result of a rotation applied on a vector / point in 3D, i.e. more calculations (about doubles compared with the Rodrigues rotation formula) are performed than necessary if the formula written in the present section are applied directly. If you need fast implementation, consider using the Rodrigues or similar formula instead of the application of a unit quaternion on a vector: `https://gamedev.stackexchange.com/questions/28395/rotating-vector3-by-a-quaternion`.