Title:
We Don't Really Need Quaternions in Geometric Modeling, Computer Graphics and Animation. Here Is Why

Authors:
Fuhua (Frank) Cheng, cheng@cs.uky.edu, University of Kentucky
T. Lee Johnson, timothy.johnsonii@uky.edu, University of Kentucky
Anastasia Kazadi, ansm226@g.uky.edu, University of Kentucky
Ethan G. Toney, ethan.toney@uky.edu, University of Kentucky
Jonathan I. Watson, jonathan.watson@uky.edu, University of Kentucky
Alice J. Lin, lina@apsu.edu, Austin Peay State University

Keywords:
Quaternion, Rotation, Geometric Modeling, Computer Graphics, Computer Animation

DOI: 10.14733/cadconfP.2021.334-338

Introduction:
It has long been believed that quaternions are more efficient to use for 3D rotations than ordinary rotation approach [10]. It is also commonly believed that the geometric meaning of quaternions is more obvious[10;15]. The justification is two-folded. First, the 3×3 matrix representation of a 3D rotation is expensive to use. For instance, to compose two rotations, one needs to compute the product of the two corresponding matrices, which requires twenty-seven multiplications and eighteen additions. Secondly, the matrix representation is redundant as only four of its nine entries are independent and it is not easy to extract information on rotation axis and rotation angle from the matrix representation of a 3D rotation. Quaternions, on the other hand, are cheaper to use. To compose two rotations, one only needs to do nineteen multiplications and seven additions. One can also easily recover rotation axis and rotation angle from the representation of a rotation quaternion.

While the above justification is indeed true for some aspects of the problem, it overlooked several important features of ordinary rotation approach and these features are actually critical in judging which technique should be used in a particular application. For instance, the matrix representation of a 3D rotation is important for processing geometric models with a large number of points/vertices. Further study shows that a special matrix representation of 3D rotation is not only more efficient in most applications involving geometric objects, but also more general than quaternion rotation when extracting rotation axis and rotation angle of a 3D rotation is concerned. Even more surprisingly, generating a smooth curve to interpolate a set of points on a 3D sphere can be done using ordinary rotation as well if one knows how to interpolate rotations on a 3D sphere. Besides, matrix represented 3D rotations can be accumulated with other transformations such as *translation, scaling*, *shearing* and *reflection* (in homogeneous coordinates) so one can accomplish all the transformations specified by the user in modeling space (plus the projection process) with only one vector-matrix multiplication. This is how we make real-time performance possible in computer graphics and computer animation. Therefore, there is no reason to use quaternions in geometric modeling, computer graphics and computer animation at all.

The rest of the paper is arranged as follows. In section 2, definitions of quaternion and quaternion rotation and properties of quaternion rotation will be reviewed, including interpolation of rotations represented by quaternions. In Section 3, we study some important properties of ordinary rotation and discuss applications of two important rotation representations. In Section 4, we discuss the relationship

between general rotation and principal rotations. In Section 5, we show that interpolation of rotations can actually be implemented using ordinary rotation as well. Concluding remarks are given in Section 6.

Quaternions and Quaternion Rotations:

We briefly review some basic properties of quaternions here [2;11].

Ordinary Rotation:

In this section we will discuss applications of two important rotation representations. The goal is to show that underneath the surface, ordinary rotation has advantages that we sometime overlook.

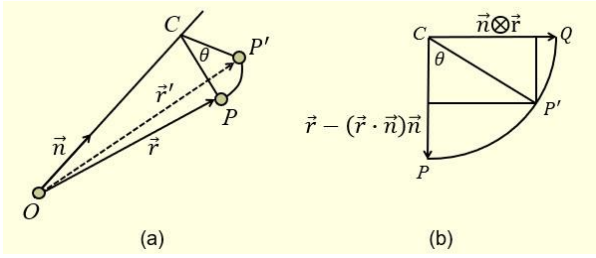

Fig. 1: Ordinary rotation.

If a vector $\vec{r} = (r_u, r_v, r_w)^t$ (or, a point $P = (P_u, P_v, P_w)^t$) is rotated about a unit vector $\hat{\boldsymbol{n}} = (\boldsymbol{n}_u, \boldsymbol{n}_v, \boldsymbol{n}_w)^t$ for θ degree (see Fig. 1), the resulting vector $\vec{r}' = (r_u', r_v', r_w')^t$ (or, point P') can be computed as follows

$$\begin{aligned}\vec{r}' &= (\vec{r}\cdot\hat{\boldsymbol{n}})\hat{\boldsymbol{n}} + cos\theta(\vec{r} - (\vec{r}\cdot\hat{\boldsymbol{n}})\hat{\boldsymbol{n}}) + sin\theta(\hat{\boldsymbol{n}}\otimes\vec{r}) \\ &= \cos\theta\vec{r} + (1\text{-}\cos\theta)(\vec{r}\cdot\hat{n})\hat{n} + \sin\theta(\hat{n}\otimes\vec{r}) \end{aligned} \quad (3\text{-}1)$$

O in Fig. 1 is the origin of the UVW-coordinate system.

It can be shown Eq. (3-1) has the following matrix form.

$$\vec{r}' = \begin{bmatrix} r_u' \\ r_v' \\ r_w' \end{bmatrix} = M_{R(\theta,\hat{n})}\vec{r} = M_{R(\theta,\hat{n})}\begin{bmatrix} r_u \\ r_v \\ r_w \end{bmatrix} \quad (3\text{-}2)$$

where $M_{R(\theta,\hat{\boldsymbol{n}})}$ is a 3×3 matrix defined as follows:

$$M_{R(\theta,\hat{\boldsymbol{n}})} = \begin{bmatrix} cos\theta + (1-cos\theta)\boldsymbol{n}_u\boldsymbol{n}_u & (1-cos\theta)\boldsymbol{n}_u\boldsymbol{n}_v - sin\theta\boldsymbol{n}_w & (1-cos\theta)\boldsymbol{n}_u\boldsymbol{n}_w + sin\theta\boldsymbol{n}_v \\ (1-cos\theta)\boldsymbol{n}_u\boldsymbol{n}_v + sin\theta\boldsymbol{n}_w & cos\theta + (1-cos\theta)\boldsymbol{n}_v\boldsymbol{n}_v & (1-cos\theta)\boldsymbol{n}_v\boldsymbol{n}_w - sin\theta\boldsymbol{n}_u \\ (1-cos\theta)\boldsymbol{n}_u\boldsymbol{n}_w - sin\theta\boldsymbol{n}_v & (1-cos\theta)\boldsymbol{n}_v\boldsymbol{n}_w + sin\theta\boldsymbol{n}_u & cos\theta + (1-cos\theta)\boldsymbol{n}_w\boldsymbol{n}_w \end{bmatrix} \quad (3\text{-}3)$$

Using eq. (3-2) for the computation of the rotation of a single point is more expensive than using quaternions because one needs to compute the matrix $M_{R(\theta,\hat{\boldsymbol{n}})}$ first which requires 24 multiplications and 10 additions/subtractions and then perform the vector-matrix multiplication which requires 9 multiplications and 6 additions, so totally one needs 33 multiplications and 16 additions/subtractions. However, if one needs to perform the same rotation for many points, such as all the vertices of the mesh representation of a car model or even just the mesh representation of a teaspoon model (100+ vertices), then eq. (3-2) is a more efficient approach to use than using quaternions because one only needs to compute the matrix $M_{R(\theta,\hat{\boldsymbol{n}})}$ once and then it can be used for the rotation of all the mesh vertices, so the total cost is the construction of the matrix $M_{R(\theta,\vec{u})}$ plus the number of vertices n times the cost of a single vector-matrix multiplication: (24+9n) multiplications + (10+6n) additions/subtractions. while the total cost of using quaternion rotation is (19n) multiplications + (7n) additions/subtractions. When n is

large, the construction cost of the matrix $M_{R(\theta,\hat{\boldsymbol{n}})}$ is a relatively small portion of the entire cost and can be ignored. Hence, when n is large, the computation cost for using quaternion rotation is (19n multiplications + 7n additions) compared to (9n multiplications + 6n additions) for eq. (3-2).

Another advantage of eq. (3-2) is certainly its capability to be accumulated with other transformations such as *translation, scaling*, *shearing* and *reflection* (in homogeneous coordinates-based representation) so that one can accomplish all the transformations specified by the user in the modeling space (plus the projection process) with only one vector-matrix multiplication. This is how we make real-time performance possible in most applications in addition to relying on hardware-implementation of the rendering algorithms.

General Rotation:
Given a principal rotation about the U-axis for θ_u degree, represented as $M_{R(\theta_u)}$, a principal rotation about the V-axis for θ_v degree, represented as $M_{R(\theta_v)}$, and a principal rotation about the W-axis for θ_w degree, represented as $M_{R(\theta_w)}$. $M_{R(\theta_u)}$, $M_{R(\theta_v)}$ and $M_{R(\theta_w)}$ can be expressed in homogeneous coordinates-based representation as follows.

$$M_{R(\theta_u)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta_u & -sin\theta_u & 0 \\ 0 & sin\theta_u & cos\theta_u & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M_{R(\theta_v)} = \begin{bmatrix} cos\theta_v & 0 & sin\theta_v & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta_v & 0 & cos\theta_v & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M_{R(\theta_w)} = \begin{bmatrix} cos\theta_w & -sin\theta_w & 0 & 0 \\ sin\theta_w & cos\theta_w & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If a point in homogeneous coordinates $(r_u, r_v, r_w, 1)^t$ is rotated about the U-axis, the V-axis and the W-axis for θ_u, θ_v, and θ_w degrees, respectively, the resulting point $(r_u',\ r_v',\ r_w',\ 1)^t$ is obtained by pre-multiplying $(r_u, r_v, r_w, 1)^t$ by the matrices $M_{R(\theta_u)}$ $M_{R(\theta_v)}$ and $M_{R(\theta_w)}$ as follows.

$$\begin{bmatrix} r_u' \\ r_v' \\ r_w' \\ 1 \end{bmatrix} = M_{R(\theta_w)} M_{R(\theta_v)} M_{R(\theta_u)} \begin{bmatrix} r_u \\ r_v \\ r_w \\ 1 \end{bmatrix} \tag{4-1}$$

It can be shown that (4-1) can actually be replaced with a single ordinary rotation. The rotation is performed about a rotation axis (unit vector) $\hat{a} = (a_u, a_v, a_w)$ defined as follows.

$$\hat{a} = \frac{(\alpha,\beta,\gamma)}{\sqrt{\alpha^2+\beta^2+\gamma^2}} \tag{4-2}$$

where

$$\begin{aligned} \alpha &= sin\left(\frac{\theta_u}{2}\right) cos\left(\frac{\theta_v}{2}\right) cos\left(\frac{\theta_w}{2}\right) - cos\left(\frac{\theta_u}{2}\right) sin\left(\frac{\theta_v}{2}\right) sin\left(\frac{\theta_w}{2}\right) \\ \beta &= cos\left(\frac{\theta_u}{2}\right) sin\left(\frac{\theta_v}{2}\right) cos\left(\frac{\theta_w}{2}\right) + sin\left(\frac{\theta_u}{2}\right) cos\left(\frac{\theta_v}{2}\right) sin\left(\frac{\theta_w}{2}\right) \\ \gamma &= cos\left(\frac{\theta_u}{2}\right) cos\left(\frac{\theta_v}{2}\right) sin\left(\frac{\theta_w}{2}\right) - sin\left(\frac{\theta_u}{2}\right) sin\left(\frac{\theta_v}{2}\right) cos\left(\frac{\theta_w}{2}\right) \end{aligned} \tag{4-3}$$

The rotation angle θ is defined through the following triangular functions

$$sin\left(\frac{\theta}{2}\right) = \sqrt{\alpha^2 + \beta^2 + \gamma^2} \tag{4-4}$$

$$cos\left(\frac{\theta}{2}\right) = cos\left(\frac{\theta_u}{2}\right) cos\left(\frac{\theta_v}{2}\right) cos\left(\frac{\theta_w}{2}\right) + sin\left(\frac{\theta_u}{2}\right) sin\left(\frac{\theta_v}{2}\right) sin\left(\frac{\theta_w}{2}\right) \tag{4-5}$$

What this says is, if one defines an ordinary rotation matrix $M_{R(\theta,\hat{a})}$ as follows

$$M_{R(\theta,\hat{a})} = \begin{bmatrix} cos(\theta)+(1-cos(\theta))a_u a_u & (1-cos(\theta))a_u a_v - sin(\theta)a_w & (1-cos(\theta))a_u a_w + sin(\theta)a_v & 0 \\ (1-cos(\theta))a_u a_v + sin(\theta)a_w & cos(\theta)+(1-cos(\theta))a_v a_v & (1-cos(\theta))a_v a_w - sin(\theta)a_u & 0 \\ (1-cos(\theta))a_u a_w - sin(\theta)a_v & (1-cos(\theta))a_v a_w + sin(\theta)a_u & cos(\theta)+(1-cos(\theta))a_w a_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4\text{-}6)$$

where θ and $\hat{a} = (a_u, a_v, a_w, 1)$ are defined as above, then if we rotate $(r_u, r_v, r_w, 1)^t$ about the rotation axis $\hat{a} = (a_u, a_v, a_w, 1)$ for θ degree, the result computed as follows is the same as the result computed using (4-1).

$$\begin{bmatrix} r_u' \\ r_v' \\ r_w' \\ 1 \end{bmatrix} = M_{R(\theta,\hat{a})} \begin{bmatrix} r_u \\ r_v \\ r_w \\ 1 \end{bmatrix}$$

This work shows that one can easily recover rotation axis and rotation angle for ordinary rotations from the representation techniques presented in Sections 3 and 4 in a more general way than quaternion rotation.

Interpolation of Rotations:

In this section, we show that to generate a closed path (space) curve C(u) on the 3D sphere S that passes through a set of given points, instead of considering unit quaternions, we can simply consider unit 3D vectors. The key here is to compute any point on a circular arc on 3D sphere S using ordinary rotation. A technique that can interpolate rotations on the 3D sphere S is introduced here. With this technique, we have the following procedure to generate a good C1-continuous curve on S that interpolates all the given 3D unit vectors without using quaternion rotation at all.

Q[0][0] = P_1;
Q[0][1] = a_1;
Q[0][2] = b_2;
Q[0][3] = P_2;
Current = Q[0][0];
$u = 0.0$;
$\Delta u = 0.01$;
for (i=0; i<100; i++) {
  $u = u + \Delta u$;
  for (j=1; j<=3; j++) {
    for (k=j; k<=3; k++) {
      cos$\theta_{j,k} = Q[j][k-1] \cdot Q[j][k]$;
      sin$\theta_{j,k} = \sqrt{1-(cos\theta_{j,k})^2}$;
      //sin$\theta_{j,k}$ is negative if cos$\theta_{j.k}$ is negative
      Q[j][k] = $\frac{sin((1-u)\theta_{j,k})}{sin\theta_{j,k}}$Q[j][k-1]+ $\frac{sin(u\theta_{j,k})}{sin\theta_{j,k}}$Q[j][k];
    }
  }
  Next = Q [3][3];
  Line (Current, Next); //Draw a line segment from Current to Next
  Current = Next;
}

Conclusions:
From the work shown in Sections 3, 4 and 5, one can see that anything quaternions can do, ordinary rotation can do as well and actually more efficiently for most of the applications in geometric modeling, computer graphics and computer animation. This is because for most applications in these areas, one usually deals with geometric models with large number of points/vertices. Therefore, the techniques presented in Section 3 is more efficient than using quaternions. Another important advantage of the representation techniques presented in Sections 3 and 4 is its capability to be accumulated with other transformations in homogeneous coordinates so that one can accomplish all the transformations in the modeling space (plus the projection process) with only one vector-matrix multiplication, an advantage quaternions cannot enjoy. The work presented in Section 4 also shows that one can easily recover rotation axis and rotation angle for ordinary rotations from the representation techniques presented in Sections 3 and 4, and actually in a more general way. Most importantly, quaternion rotation commonly used in generating smooth curves to interpolate a set of given points on 3D sphere S can be completely replaced with ordinary rotation if a technique to interpolate rotations on 3D sphere S is used. Hence, quaternions are not really needed in geometric modeling, computer graphics and computer animation.

Acknowledgement: The work of the first author is supported by NNSFC (GRANT NO. 61572020).

References:
[1] Besl, P. J.; McKay, N. D.: A method for registration of 3-D shapes, IEEE Transactions on pattern analysis and machine intelligence, 14(2), 1992, 239–256.
[2] Clifford, W. K.: Preliminary sketch of bi-quaternions, Proceedings of the London Mathematical Society, s1–4(1), 1873, 381–395.
[3] Dam, E. B.; Koch, M.; Lillholm, M.: Quaternions, Interpolation, and Animation. Technical Report DIKU-TR-98/5, Department of Computer Science, University of Copenhagen, Denmark, July 17, 1998.
[4] Euler, Leonhard: Decouverte d'un nouveau principe de mechanique, Opera omnia (1957), Ser. Secunda (Vol. 5):81{108, 1752}, Orell Fusli Turici.
[5] Faugeras, O. D.; Hebert, M.: The representation, recognition, and locating of 3-D objects, International Journal of Robotics Research, 5(3), 1986, 27–52.
[6] Hamilton, W. R.: On quaternions; or on a new system of imaginaries in algebra, London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 25(3), 1844, 489–495.
[7] Horn, B. K. P.: Closed-form solution of absolute orientation using unit quaternions, Journal of Optical Society of America A, 4(4), 1987, 629–642.
[8] Hungerford, T. W.: Algebra. Springer-Verlag, 1974.
[9] Jacobson, N.: Basic Algebra, W. H. Freeman & Co.,1985.
[10] Jia, Y.-B.: Quaternions and Rotation (Computer Science 477/577 Notes), Department of Computer Science, Iowa State University. http://web.cs.iastate.edu/~cs577/handouts/quaternion.pdf
[11] Kantor, I. L.; Solodovnikov, A.S.: Hypercomplex Numbers, An Elementary Introduction to Algebras, Springer-Verlag, 1989.
[12] Kuipers, J. B.: Quaternions and Rotation Sequences, Princeton University Press, 1999.
[13] Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P.: Numerical Recipes in C, 2nd Edition, Cambridge University Press, Inc., 2002.
[14] Schwartz, J. T.; Sharir, M.: Identification of partially obscured objects in two and three dimensions by matching noisy characteristic curves, International Journal of Robotics Research, 6(2), 1987, 29–44.
[15] Shoemake, K.: Animating rotation with quaternion curves, Computer Graphics, 19(3), 1985, 245 254.
[16] Zhao, F.; Wachem, B. G. M. van: A novel quaternion integration approach for describing the behavior of non-spherical particles, Acta Mechanica, 224, 2013, 3091–3109.