









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

Fuhua (Frank) Cheng¹ , T. Lee Johnson² , Anastasia Kazadi³ , Ethan G. Toney⁴ , Jonathan I. Watson⁵  and Alice J. Lin⁶ 

¹University of Kentucky, cheng@cs.uky.edu

²University of Kentucky, timothy.johnsonii@uky.edu

³University of Kentucky, ansm226@q.uky.edu

⁴University of Kentucky, ethan.toney@uky.edu

⁵University of Kentucky, jonathan.watson@uky.edu

⁶Austin Peay State University, lina@apsu.edu

Corresponding author: Fuhua (Frank) Cheng, cheng@cs.uky.edu

Abstract. It has long been believed that quaternions are more efficient to use for 3D rotations than ordinary rotation approach. It is also commonly believed that the geometric meaning of quaternions is more obvious. One can also easily recover rotation axis and rotation angle from the representation of a rotation quaternion. In this paper we present important features of ordinary rotation that are critical in judging which technique should be used in a particular application. We show that everything quaternion rotation can do, ordinary rotation can do as well and, actually, more efficiently, including interpolation of rotations.

Keywords: Quaternion, Rotation, Computer Graphics, Computer Animation

DOI: <https://doi.org/10.14733/cadaps.2023.1061-1073>

1 INTRODUCTION

It has long been believed that quaternions are more efficient to use for 3D rotations than ordinary rotation approach. It is also commonly believed that the geometric meaning of quaternions is more obvious. 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 that 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 are 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.

2 QUATERNIONS AND QUATERNION ROTATIONS

We briefly review some basic properties of quaternions first.

2.1 Definitions

A *quaternion* q is the combination of a scalar and a three-dimensional vector, as was originally defined by W. R. Hamilton [6]. A *quaternion* can be represented in several forms. We use the following form here:

$$q \equiv [w, \hat{v}] \quad (2-1)$$

where w is a scalar and $\hat{v} = (x, y, z)$ is a three-dimensional vector. The set of all quaternions is called S^3 . S^3 includes R as a subset in the sense that each $w \in R$, the set of real numbers, corresponds to $[w, \hat{0}] = [w, (0, 0, 0)]$ in S^3 . A quaternion $q = [w, \hat{v}]$ will simply be regarded as a scalar w if \hat{v} is a zero vector. Actually S^3 includes R^3 , the set of three-dimensional points/vectors, as a subset as well. Each point P or vector \hat{v} in R^3 corresponds to $[0, P]$ or $[0, \hat{v}]$ in S^3 . A quaternion will be regarded as a point or vector in R^3 if the scalar component of the quaternion is zero.

Given two *quaternions* $q_1 = [w_1, \hat{v}_1]$ and $q_2 = [w_2, \hat{v}_2]$ where $\hat{v}_i = (x_i, y_i, z_i), i = 1, 2$, *addition* and *subtraction* of q_1 and q_2 are defined as follows:

$$q_1 \pm q_2 \equiv [w_1 \pm w_2, \hat{v}_1 \pm \hat{v}_2] = [w_1 \pm w_2, (x_1 \pm x_2, y_1 \pm y_2, z_1 \pm z_2)]. \quad (2-2)$$

$q_1 + q_2$ and $q_1 - q_2$ are called the sum and difference of q_1 and q_2 , respectively. *Multiplication* of q_1 and q_2 is defined as

$$q_1 q_2 \equiv [w_1 w_2 - \hat{v}_1 \cdot \hat{v}_2, w_1 \hat{v}_2 + w_2 \hat{v}_1 + \hat{v}_1 \otimes \hat{v}_2] \quad (2-3)$$

where $\hat{v}_1 \cdot \hat{v}_2$ and $\hat{v}_1 \otimes \hat{v}_2$ are inner and cross products of three-dimensional vectors, respectively. $q_1 q_2$ is called the *product* of q_1 and q_2 . Quaternion multiplication is not commutative, i.e., in general, $q_1 q_2 \neq q_2 q_1$. Quaternion multiplication is associative, i.e., $(q_1 q_2) q_3 = q_1 (q_2 q_3)$ for any three quaternions q_1, q_2 and q_3 .

The *conjugate* q^* of a quaternion $q = [w, \hat{v}] = [w, (x, y, z)]$ is defined as $q^* \equiv [w, -\hat{v}] = [w, (-x, -y, -z)]$. It is easy to see that

$$qq^* = q^*q = |q|^2 = w^2 + x^2 + y^2 + z^2.$$

$|q|$ is called the *norm* of q . Quaternion norm is multiplication-invariant, i.e., $|q_1q_2| = |q_1||q_2|$ for any two quaternions q_1 and q_2 . A quaternion is called a *unit quaternion* if its norm equals one. We use S^2 to represent the set of all unit quaternions in S^3 . Each unit quaternion can be arranged in the form of $[\cos \theta, \sin \theta \hat{n}]$ for some $0 \leq \theta \leq \pi$ and some unit vector \hat{n} in R^3 .

The *inverse* q^{-1} of a quaternion $q = [w, \hat{v}] = [w, (x, y, z)]$ satisfies the condition $qq^{-1} = 1$ and can be expressed as

$$q^{-1} = \frac{q^*}{qq^*} = \frac{q^*}{|q|^2} = \frac{q^*}{w^2 + x^2 + y^2 + z^2}. \quad (2-4)$$

Multiplication of a quaternion by its inverse does not depend on the order of the multiplication. It is easy to verify that $q^{-1}q = 1$. A unit quaternion's inverse is just its conjugate.

Division of quaternion q_1 by quaternion q_2 is defined as:

$$q_1/q_2 \equiv q_1q_2^{-1} \quad (2-5)$$

where q_2^{-1} is the *inverse* of q_2 as defined in (2-4). Note that $(q_1q_2)/q_2 = (q_1q_2)q_2^{-1} = q_1(q_2q_2^{-1}) = q_1$. Hence quaternion division is an inverse operation of quaternion multiplication.

2.2 Quaternion Rotation

Let $q = [\cos(\frac{\theta_u}{2}), \sin(\frac{\theta_u}{2})(1, 0, 0)]$, q^* being its conjugate $q^* = [\cos(\frac{\theta_u}{2}), -\sin(\frac{\theta_u}{2})(1, 0, 0)]$, and $\vec{r} = (r_u, r_v, r_w)$, a unit 3D vector. A quaternion rotation about a unit vector in U-direction for θ_u degree is defined and computed as follows:

$$\begin{aligned} q * [0, \vec{r}] * q^* &= \left[\cos\left(\frac{\theta_u}{2}\right), \sin\left(\frac{\theta_u}{2}\right)(1, 0, 0) \right] * [0, \vec{r}] * \left[\cos\left(\frac{\theta_u}{2}\right), -\sin\left(\frac{\theta_u}{2}\right)(1, 0, 0) \right] \\ &= \left[-\sin\left(\frac{\theta_u}{2}\right)r_u, \cos\left(\frac{\theta_u}{2}\right)\vec{r} + \sin\left(\frac{\theta_u}{2}\right)(1, 0, 0) \otimes \vec{r} \right] * \left[\cos\left(\frac{\theta_u}{2}\right), -\sin\left(\frac{\theta_u}{2}\right)(1, 0, 0) \right] \\ &= \left[-\sin\left(\frac{\theta_u}{2}\right)r_u, \cos\left(\frac{\theta_u}{2}\right)\vec{r} + \sin\left(\frac{\theta_u}{2}\right)(0, -r_w, r_v) \right] * \left[\cos\left(\frac{\theta_u}{2}\right), -\sin\left(\frac{\theta_u}{2}\right)(1, 0, 0) \right] \\ &= \left[-\sin\left(\frac{\theta_u}{2}\right)\cos\left(\frac{\theta_u}{2}\right)r_u + \cos\left(\frac{\theta_u}{2}\right)\sin\left(\frac{\theta_u}{2}\right)r_u, \right. \\ &\quad \left. \cos^2\left(\frac{\theta_u}{2}\right)\vec{r} + \cos\left(\frac{\theta_u}{2}\right)\sin\left(\frac{\theta_u}{2}\right)(0, -r_w, r_v) + \sin^2\left(\frac{\theta_u}{2}\right)(r_u, 0, 0) \right. \\ &\quad \left. - \cos\left(\frac{\theta_u}{2}\right)\sin\left(\frac{\theta_u}{2}\right)\vec{r} \otimes (1, 0, 0) - \sin^2\left(\frac{\theta_u}{2}\right)(0, -r_w, r_v) \otimes (1, 0, 0) \right] \\ &= [0, \cos^2\left(\frac{\theta_u}{2}\right)\vec{r} + 2\cos\left(\frac{\theta_u}{2}\right)\sin\left(\frac{\theta_u}{2}\right)(0, -r_w, r_v) + \sin^2\left(\frac{\theta_u}{2}\right)(r_u, -r_v, -r_w)] \end{aligned}$$

$$= [0, (r_u \cos \theta_u r_v - \sin \theta_v r_w, \cos \theta_u r_w + \sin \theta_u r_v)] \quad (2-6)$$

3 ORDINARY ROTATION

In this section we will review and re-visit some important properties of ordinary rotation and discuss applications of two important rotation representations. The goal is to show that underneath the surface ordinary rotation has advantages that we sometime overlook.

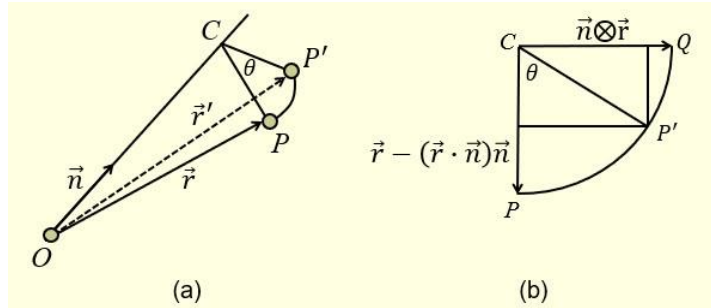


Figure 1: Rotation of a vector \vec{r} about a unit vector \vec{n} for θ degree.

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 $\hat{n} = (n_u, n_v, n_w)^t$ 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

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

O in Fig. 1 is the origin of the UVW-coordinate system. Since the three terms in the second line of Eq. (3-1) satisfy the following properties,

$$\begin{aligned} \cos\theta\vec{r} &= \cos\theta \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_u \\ r_v \\ r_w \end{bmatrix}, \quad (1 - \cos\theta)(\vec{r} \cdot \hat{n})\hat{n} = (1 - \cos\theta) \begin{bmatrix} n_u n_u & n_v n_u & n_w n_u \\ n_u n_v & n_v n_v & n_w n_v \\ n_u n_w & n_v n_w & n_w n_w \end{bmatrix} \begin{bmatrix} r_u \\ r_v \\ r_w \end{bmatrix}, \\ \sin\theta(\hat{n} \otimes \vec{r}) &= \sin\theta \begin{bmatrix} 0 & -n_w & n_v \\ n_w & 0 & -n_u \\ -n_v & n_u & 0 \end{bmatrix} \begin{bmatrix} r_u \\ r_v \\ r_w \end{bmatrix} \end{aligned}$$

By substituting the above equations into (3-1) and combining corresponding entries, we have the following matrix form of Eq. (3-1).

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

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

$$M_{R(\theta, \hat{n})} = \begin{bmatrix} \cos\theta + (1 - \cos\theta)n_u n_u & (1 - \cos\theta)n_u n_v - \sin\theta n_w & (1 - \cos\theta)n_u n_w + \sin\theta n_v \\ (1 - \cos\theta)n_u n_v + \sin\theta n_w & \cos\theta + (1 - \cos\theta)n_v n_v & (1 - \cos\theta)n_v n_w - \sin\theta n_u \\ (1 - \cos\theta)n_u n_w - \sin\theta n_v & (1 - \cos\theta)n_v n_w + \sin\theta n_u & \cos\theta + (1 - \cos\theta)n_w n_w \end{bmatrix} \quad (3-3)$$

Using eq. (3-2) for the computation of the rotation of a single point is more expensive than eq. (2-6) because one needs to compute the matrix $M_{R(\theta, \hat{n})}$ 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 eq. (2-6) because one only needs to compute the matrix $M_{R(\theta, \hat{n})}$ 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 $M_{R(\theta, \hat{n})}$ 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 for eq. (2-6) is $(19n)$ multiplications + $(7n)$ additions/subtractions

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

4 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.

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

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

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

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.

$$\begin{bmatrix} r'_u \\ r'_v \\ r'_w \\ 1 \end{bmatrix} = M_{R(\theta_w)} M_{R(\theta_v)} M_{R(\theta_u)} \begin{bmatrix} r_u \\ r_v \\ r_w \\ 1 \end{bmatrix} \quad (4-1)$$

Note that

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

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

Hence, $M_{R(\theta_w)}M_{R(\theta_v)}M_{R(\theta_u)}$ can be expressed as

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

$$= \begin{bmatrix} \cos(\theta_v)\cos(\theta_w) & \sin(\theta_u)\sin(\theta_v)\cos(\theta_w) - \cos(\theta_u)\sin(\theta_w) & \cos(\theta_u)\sin(\theta_v)\cos(\theta_w) + \sin(\theta_u)\sin(\theta_w) & 0 \\ \cos(\theta_v)\sin(\theta_w) & \sin(\theta_u)\sin(\theta_v)\sin(\theta_w) + \cos(\theta_u)\cos(\theta_w) & \cos(\theta_u)\sin(\theta_v)\sin(\theta_w) - \sin(\theta_u)\cos(\theta_w) & 0 \\ -\sin(\theta_v) & \sin(\theta_u)\cos(\theta_v) & \cos(\theta_u)\cos(\theta_v) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4-2)$$

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

$$\hat{a} = \frac{(\alpha, \beta, \gamma)}{\sqrt{\alpha^2 + \beta^2 + \gamma^2}} \quad (4-3)$$

where

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

The rotation angle θ is defined through the following trigonometric functions

$$\sin\left(\frac{\theta}{2}\right) = \sqrt{\alpha^2 + \beta^2 + \gamma^2} \quad (4-5)$$

$$\cos\left(\frac{\theta}{2}\right) = \cos\left(\frac{\theta_u}{2}\right)\cos\left(\frac{\theta_v}{2}\right)\cos\left(\frac{\theta_w}{2}\right) + \sin\left(\frac{\theta_u}{2}\right)\sin\left(\frac{\theta_v}{2}\right)\sin\left(\frac{\theta_w}{2}\right) \quad (4-6)$$

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

$$M_{R(\theta, \hat{a})} = \begin{bmatrix} \cos(\theta) + (1 - \cos(\theta))a_u a_u & (1 - \cos(\theta))a_u a_v - \sin(\theta)a_w & (1 - \cos(\theta))a_u a_w + \sin(\theta)a_v & 0 \\ (1 - \cos(\theta))a_u a_v + \sin(\theta)a_w & \cos(\theta) + (1 - \cos(\theta))a_v a_v & (1 - \cos(\theta))a_v a_w - \sin(\theta)a_u & 0 \\ (1 - \cos(\theta))a_u a_w - \sin(\theta)a_v & (1 - \cos(\theta))a_v a_w + \sin(\theta)a_u & \cos(\theta) + (1 - \cos(\theta))a_w a_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4-7)$$

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.

5 INTERPOLATION OF ROTATIONS

Our task here is to generate a closed path (space) curve $C(u)$ on the 3D sphere S that passes through a set of given points P_0, P_1, \dots, P_n on S . $C(u)$ will be at least C^1 -continuous. For this task, instead of considering q_1 and q_2 on S^3 (the set of unit quaternions), we can simply consider \hat{v}_1 and \hat{v}_2 on S^2 (the set of unit 3D vectors).

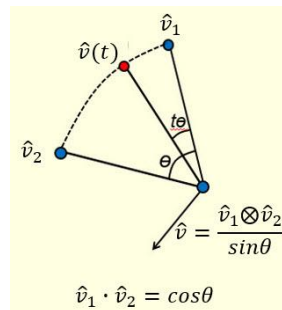


Figure 2: Construction of the rotation axis \hat{v} .

For two given unit vectors $\hat{v}_1 = (\hat{v}_{1x}, \hat{v}_{1y}, \hat{v}_{1z})^t$ and $\hat{v}_2 = (\hat{v}_{2x}, \hat{v}_{2y}, \hat{v}_{2z})^t$ in S^2 , define \hat{v} as follows (see Fig. 2)

$$\hat{v} = \frac{\hat{v}_1 \otimes \hat{v}_2}{\sin \theta} \quad (5-1)$$

Then any point on the circular arc between \hat{v}_1 and \hat{v}_2 can be computed as follows

$$\hat{v}(t) = \frac{\sin((1-t)\theta)}{\sin \theta} \hat{v}_1 + \frac{\sin(t\theta)}{\sin \theta} \hat{v}_2 \quad (5-2)$$

where $0 \leq t \leq 1$ and $\hat{v}_1 \cdot \hat{v}_2 = \cos \theta$.

Instead of eq. (5-2), a second choice to compute $\hat{v}(t)$ is to use the following formula

$$\begin{bmatrix} \hat{v}_x(t) \\ \hat{v}_y(t) \\ \hat{v}_z(t) \\ 1 \end{bmatrix} = M_{R(t\theta, \hat{v})} \begin{bmatrix} \hat{v}_{1x} \\ \hat{v}_{1y} \\ \hat{v}_{1z} \\ 1 \end{bmatrix} \quad (5-3)$$

where $\hat{v}_x(t)$, $\hat{v}_y(t)$ and $\hat{v}_z(t)$ are x, y and z components of $\hat{v}(t)$ and $M_{R(t\theta, \hat{v})}$ is a 4×4 rotation matrix defined in (3-3). Note that the *slerp* technique cannot have a vector-matrix multiplication representation like eq. (5-3).

Eq. (5-2) or eq. (5-3) can only give us a circular arc between two consecutive points P_{i-1} and P_i . We need to use 3D cubic composite Bezier curve technique to construct a smooth path curve to interpolate the given points P_0, P_1, \dots, P_n , and use eq. (5-2) or eq. (5-3) only in the computation of the circular control polygon of the path curve on S^2 .

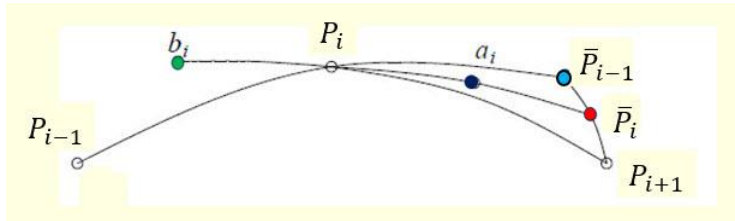


Figure 3: Construction of the control points a_i and b_i .

To compute the control points of the cubic composite Bezier curve that interpolates the given points P_0, P_1, \dots, P_n , we use Shoemake's 2nd approach to compute a_i and b_i for each set of three consecutive points P_{i-1}, P_i , and P_{i+1} (see Fig. 3).

$$a_i = \text{Bisect}(P_i, \text{Bisect}(\text{Double}(P_{i-1}, P_i), P_{i+1}))$$

$$b_i = \text{Double}(a_i, P_i)$$

See Fig. 3 for the locations of a_i and b_i . In this figure, $\bar{P}_{i-1} = \text{Double}(P_{i-1}, P_i)$, $\bar{P}_i = \text{Bisect}(\bar{P}_{i-1}, P_{i+1})$. If the angle between P_{i-1} and P_i is θ_{i-1} then the angle between P_{i-1} and \bar{P}_{i-1} is $2\theta_{i-1}$ (see Fig. 4). Hence, \bar{P}_{i-1} can be computed as follows:

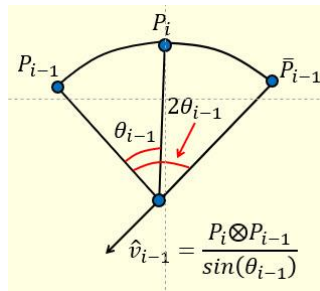


Figure 4: Construction of the rotation axis \hat{v}_{i-1} .

$$\bar{P}_{i-1} = 2\cos(\theta_{i-1})P_i - P_{i-1} \quad (5-4)$$

or

$$\begin{bmatrix} \bar{P}_{(i-1)x} \\ \bar{P}_{(i-1)y} \\ \bar{P}_{(i-1)z} \\ 1 \end{bmatrix} = M_{R(2\theta_{i-1}, \hat{v}_{i-1})} \begin{bmatrix} P_{(i-1)x} \\ P_{(i-1)y} \\ P_{(i-1)z} \\ 1 \end{bmatrix} \quad (5-5)$$

where $M_{R(2\theta_{i-1}, \hat{v}_{i-1})}$ is a 4×4 rotation matrix defined in (3-3) and \hat{v}_{i-1} is the rotation axis defined as follows

$$\hat{v}_{i-1} = \frac{P_{i-1} \otimes P_i}{\sin(\theta_{i-1})} \quad (5-6)$$

with θ_{i-1} being the angle between P_{i-1} and P_i . Our path curve is a closed curve, therefore, $i-1$ and $i+1$ are modulo n .

Once we have \bar{P}_{i-1} , we can compute \bar{P}_i as follows

$$\bar{P}_i = \frac{\sin(\bar{\theta}_{i-1}/2)}{\sin(\theta_{i-1})} P_{i+1} + \frac{\sin(\bar{\theta}_{i-1}/2)}{\sin(\theta_{i-1})} \bar{P}_{i-1} \quad (5-7)$$

or

$$\begin{bmatrix} \bar{P}_{ix} \\ \bar{P}_{iy} \\ \bar{P}_{iz} \\ 1 \end{bmatrix} = M_{R(\bar{\theta}_{i-1}/2, \hat{v}_{i-1})} \begin{bmatrix} \bar{P}_{(i-1)x} \\ \bar{P}_{(i-1)y} \\ \bar{P}_{(i-1)z} \\ 1 \end{bmatrix} \quad (5-8)$$

where $M_{R(\bar{\theta}_{i-1}/2, \hat{v}_{i-1})}$ is a 4×4 rotation matrix defined in (3-3) with $\bar{\theta}_{i-1}$ being the angle between \bar{P}_{i-1} and P_{i+1} , and \hat{v}_{i-1} being the rotation axis defined as follows

$$\hat{v}_{i-1} = \frac{P_{i+1} \otimes \bar{P}_{i-1}}{\sin(\theta_{i-1})} \quad (5-9)$$

Figure 5: Construction of the rotation axis \hat{v}_{i-1} .

We are ready to compute a_i and b_i now. $a_i = \text{Bisect}(P_i, \bar{P}_i)$, mid-point of the circular arc between P_i and \bar{P}_i , is computed as follows

$$a_i = \frac{\sin(\bar{\theta}_i/2)}{\sin(\theta_i)} P_i + \frac{\sin(\bar{\theta}_i/2)}{\sin(\theta_i)} \bar{P}_i \quad (5-10)$$

or

$$\begin{bmatrix} a_{ix} \\ a_{iy} \\ a_{iz} \\ 1 \end{bmatrix} = M_{R(\bar{\theta}_i/2, \hat{v}_i)} \begin{bmatrix} \bar{P}_{ix} \\ \bar{P}_{iy} \\ \bar{P}_{iz} \\ 1 \end{bmatrix} \quad (5-11)$$

where $M_{R(\bar{\theta}_i/2, \hat{v}_i)}$ is a 4×4 rotation matrix defined in (3-3) with $\bar{\theta}_i$ being the angle between \bar{P}_i and P_i (see Fig. 6) and \hat{v}_i being the rotation axis defined as follows

$$\hat{v}_i = \frac{\bar{P}_i \otimes P_i}{\sin(\theta_i)} \quad (5-12)$$

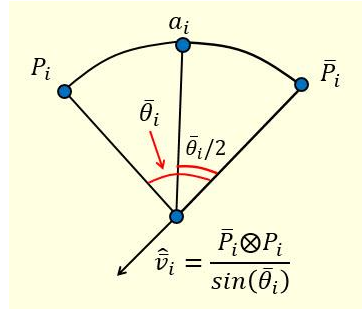


Figure 6: Construction of the rotation axis \hat{v}_i .

$b_i = \text{Double}(a_i, P_i)$ is obtained by extending the circular arc $\widehat{a_i P_i}$ in the direction of $\overrightarrow{a_i P_i}$ so that the angle between b_i and P_i is the same as the angle between P_i and a_i (see Fig. 7).

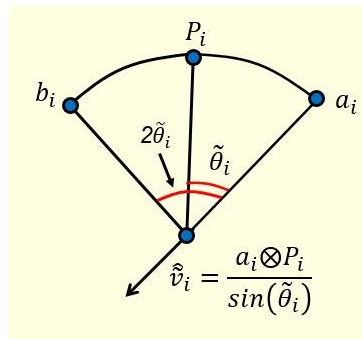


Figure 7: Construction of the rotation axis \hat{v}_i .

b_i is computed as follows

$$b_i = 2\cos(\tilde{\theta}_i)a_i - P_i \quad (5-13)$$

or

$$\begin{bmatrix} b_{ix} \\ b_{iy} \\ b_{iz} \\ 1 \end{bmatrix} = M_{R(2\tilde{\theta}_i, \hat{v}_i)} \begin{bmatrix} a_{ix} \\ a_{iy} \\ a_{iz} \\ 1 \end{bmatrix} \quad (5-14)$$

where $M_{R(2\tilde{\theta}_i, \hat{v}_i)}$ is a 4×4 rotation matrix defined in (3-3) with $\tilde{\theta}_i$ being the angle between a_i and P_i (see Fig. 7) and \hat{v}_i being the rotation axis defined as follows

$$\hat{v}_i = \frac{a_i \otimes P_i}{\sin(\tilde{\theta}_i)} \quad (5-15)$$

Once we have all the a_i and b_i constructed, we are ready to construct a closed composite cubic Bezier curve on S^2 that interpolates all the P_i . Fig. 8 shows the relationship between a_i , b_i , P_i and the composite cubic Bezier curve $C(u)$. Each segment of $C(u)$ is defined by four control points. For instance, the second segment of $C(u)$, denoted $C_2(u)$, is defined by control points P_1 , a_1 , b_2 and P_2 . In general, the i -th segment $C_i(u)$ is defined by control points P_{i-1} , a_{i-1} , b_i and P_i .

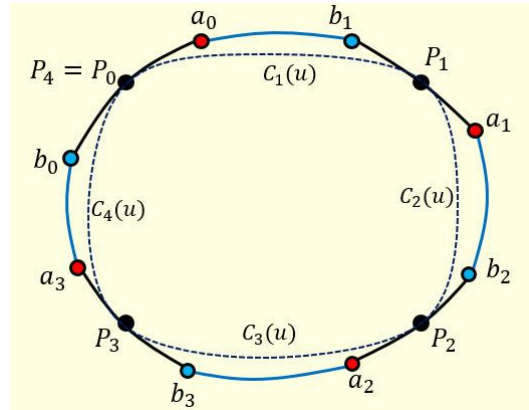


Figure 8: A closed composite cubic Bezier curve that interpolates P_0, P_1, P_2, P_3 and $P_4 (= P_0)$ on S^2 .

The parameter space of each segment is the same: the unit interval $[0, 1]$. Each segment of the curve is generated using the *de Casteljau* algorithm for circular arc interpolation instead of linear segment interpolation. For example, for the second curve segment $C_2(u)$, if 100 points are to be generated for the curve segment, we set step size to be $\Delta u = 0.01$ and then use the following pseudo code to generate the curve segment:

```

Q[0][0] = P1;
Q[0][1] = a1;
Q[0][2] = b2;
Q[0][3] = P2;
Current = Q[0][0];
u = 0.0;
Δu = 0.01;
for (i=0; i<100; i++) {
    u = u + Δu;
    for (j=1; j<=3; j++) {
        for (k=j; k<=3; k++) {
            cosθj,k = Q[j][k-1] · Q[j][k];
            sinθj,k = √(1 - (cosθj,k)2);
            //sinθj,k is negative if cosθj,k is negative
            Q[j][k] = (sin((1-u)θj,k)/sinθj,k)Q[j][k-1] + (sin(uθj,k)/sinθj,k)Q[j][k];
        }
    }
    Next = Q[3][3];
    Line(Current, Next); //Draw a line segment from Current to Next
    Current = Next;
}

```

This procedure generates a good C^1 -continuous curve on S^2 that interpolates all the given P_i .

6 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 Section 3 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 quaternion cannot enjoy. The work presented in Section 4 also shows that one can easily recover rotation axis and rotation angle for ordinary rotations in a more general way than quaternion rotation. 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 developed in Section 5 is used. Hence, quaternions are not really needed in geometric modeling, computer graphics and computer animation.

7 ACKNOWLEDGEMENT

Research work of the first author is supported by the National Science Foundation of China (61572020).

8 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):239–256, 1992. <https://doi.org/10.1109/34.121791>
- [2] Clifford, W. K.: Preliminary sketch of bi-quaternions, *Proceedings of the London Mathematical Society*, s1-4(1):381–395, 1873. <https://doi.org/10.1093/nq/s4-XI.280.381c>
- [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 mecanique*, 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):27–52, 1986. <https://doi.org/10.1177/027836498600500302>
- [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):489–495, 1844. <https://doi.org/10.1080/14786444408645047>
- [7] Horn, B. K. P.: Closed-form solution of absolute orientation using unit quaternions, *Journal of Optical Society of America A*, 4(4):629–642, 1987. <https://doi.org/10.1364/JOSAA.4.000629>
- [8] 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>
- [9] Kantor, I. L.; Solodovnikov, A.S.: *Hypercomplex Numbers, An Elementary Introduction to Algebras*, Springer-Verlag, 1989.
- [10] Kuipers, J. B.: *Quaternions and Rotation Sequences*, Princeton University Press, 1999. <https://doi.org/10.1515/9780691211701>
- [11] Miura, K. T.; Wang, L.; Cheng, F.: Streamline modeling with subdivision surfaces on the Gaussian sphere, *Computer-Aided Design* 33: 975–987, 2001. [https://doi.org/10.1016/S0010-4485\(00\)00134-2](https://doi.org/10.1016/S0010-4485(00)00134-2)
- [12] Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P.: *Numerical Recipes in C*, 2nd Edition, Cambridge University Press, Inc., 2002.
- [13] 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):29–44, 1987. [https://doi.org/10.1016/0950-5849\(87\)90020-6](https://doi.org/10.1016/0950-5849(87)90020-6)
- [14] Shoemake, K.: Animating rotation with quaternion curves, *Computer Graphics*, 19(3):245–254, 1985. <https://doi.org/10.1145/325165.325242>

- [15] Zhao, F.; Wachem, B. G. M. van: A novel quaternion integration approach for describing the behavior of non-spherical particles, *Acta Mechanica*, 224:3091–3109, 2013.
<https://doi.org/10.1007/s00707-013-0914-2>