Int. J. Appl. Math. Comput. Sci., 2020, Vol. 30, No. 1, 149–160
DOI: 10.34768/amcs-2020-0012

AN ALGORITHM FOR QUATERNION–BASED 3D ROTATION

ALEKSANDR CARIOW [a], GALINA CARIOWA [a], DOROTA MAJORKOWSKA-MECH [a,*]

[a]Faculty of Computer Science and Information Technology
West Pomeranian University of Technology in Szczecin
ul. Żołnierska 49, 71-210 Szczecin, Poland
e-mail: dmajorkowska@wi.zut.edu.pl

In this work a new algorithm for quaternion-based spatial rotation is presented which reduces the number of underlying real multiplications. The performing of a quaternion-based rotation using a rotation matrix takes 15 ordinary multiplications, 6 trivial multiplications by 2 (left-shifts), 21 additions, and 4 squarings of real numbers, while the proposed algorithm can compute the same result in only 14 real multiplications (or multipliers—in a hardware implementation case), 43 additions, 4 right-shifts (multiplications by 1/4), and 3 left-shifts (multiplications by 2).

Keywords: quaternions, space rotation, design of algorithms.

1. Introduction

Currently, quaternions (Kantor and Solodovnikov, 1989) are widely used for data processing in various fields of science and engineering including navigation (Fenwick, 1992), orbital mechanics of satellites (Kuipers, 1999; Andreis and Canuto, 2004), robotics and computer vision (Hu *et al.*, 2006; Çakir and Bütün, 2010; Wareham *et al.*, 2005; Terzakis *et al.*, 2014), autonomous vehicles (Roberts *et al.*, 2006; Fresk and Nikolakopoulos, 2013), digital signal and image processing (Bülow and Sommer, 2001; Sangwine and Bihan, 2007; Alfsmann *et al.*, 2007; Pei *et al.*, 2001; 2011; Ell, 1993; Witten and Shragge, 2006; Bayro-Corrochano, 2006), 3D game programming and computer graphics (Pletinckx, 1989; Nüchter, 2009; Angel and Shreiner, 2012; Abbena *et al.*, 2006; Mukundan, 2002; Markley, 2008; Shuster and Natanson, 1993; Vince, 2011; Lengyel, 2011; Hanson, 2006), wireless communications (Wysocki *et al.*, 2006), public-key cryptography (Malekian *et al.*, 2011), crystallographic texture analysis (Kunze and Schaeben, 2004), etc.

The most important property of quaternions is the fact that every unit quaternion represents a rotation of objects in three dimensions (3D). This plays an essential role in applying quaternions to perform 3D rotations in various machine vision applications (Rousseau *et al.*, 2002; Terzakis *et al.*, 2014; Kanade, 1987). Using unit quaternions is a most natural, elegant, and practical way to describe rotation in three-dimensional vector spaces (Çakir and Bütün, 2010; Funda *et al.*, 1990; Huynh, 2009). However, the direct applying of quaternions to rotation in three dimensions requires a relatively large amount of computation. For that reason this operation is usually implemented via multiplication of a vector by the so-called “rotation matrix”, which is simply calculated based on the coefficients of the quaternion (Markley, 2008; Shuster and Natanson, 1993). Nevertheless, even this way requires a relatively large amount of computations. In some applications, reducing the number of multiplications is especially important. In this article we will show a new algorithm that requires fewer multiplications than the one that uses a rotation matrix.

The paper is organized in the following way. Section 2 describes a rotation in 3D space and introduces a rotation matrix. Section 3 introduces quaternions and describes a rotation in 3D space using quaternions. It also explains a representation of quaternions as 4×4 matrices and clarifies the description of rotation in such a representation. Section 4 details the transformation and the derivations of the final algorithm that turns the vector into the rotation outcome. Section 5 compares the computational complexity of the presented algorithm with other methods of description of rotation in 3D space.

*Corresponding author

The last section summarizes the introduced solution and describes possible applications of the presented algorithm.

2. Rotations in three-dimensional space and the rotation matrix

Rotation is a concept coming from geometry and it is also a basic kind of motion in physics. Any rotation is a motion of a certain space that preserves at least one point. It can describe, for example, the motion of a rigid body around a fixed point. A rotation is different from other types of motion: translations, which have no fixed points, and reflections, each of them having an entire two-dimensional flat of fixed points in three-dimensional space. In geometry, Euler's rotation theorem states that, in 3D, any displacement of a rigid body such that a point on the rigid body remains fixed is equivalent to a single rotation about some axis that runs through the fixed point. It also means that the composition of two rotations is also a rotation.

Generally rotations are not commutative. The axis of rotation is known as an Euler axis, typically represented by a unit vector $\mathbf{n}$. Let P be chosen point of a rigid body, which rotates according to the right hand rule by the angle φ about the rotation axis, represented by the unit vector $\mathbf{n}$. Let a reference frame be connected with a point O, which lies on the rotation axis. In this reference frame, point P is represented by the vector $\mathbf{x}$, as shown in Fig. 1. When we

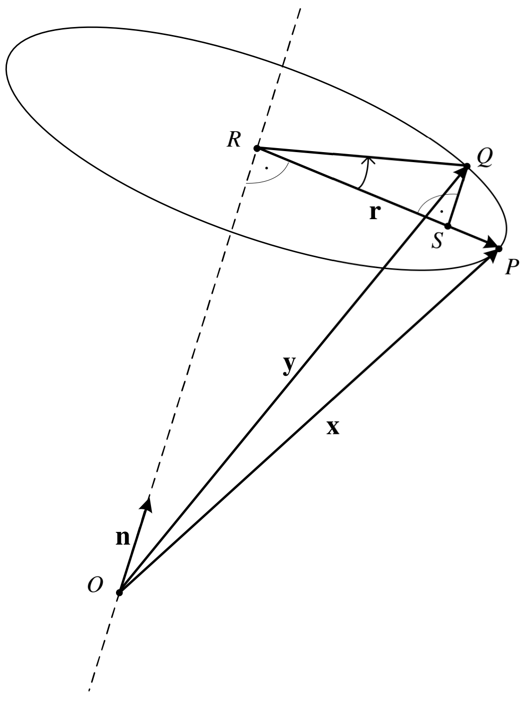

Fig. 1. Illustration of rotation of the vector $\mathbf{x}$ by the angle φ around the rotation axis represented by the unit vector $\mathbf{n}$, according to the right hand rule.

rotate the point P about the rotation axis, we get the point Q—an image of P, which is represented by the vector $\mathbf{y}$. Let the point R be an orthogonal projection of the point P onto the rotation axis and the point S be an orthogonal projection of the point Q onto the vector $\mathbf{r}$ which starts at point R and ends at point P. The vectors $\overrightarrow{RP}$ and $\overrightarrow{RQ}$ are equal in length $|RP| = |RQ| = r$. It is easy to see that

$$\overrightarrow{OQ} = \overrightarrow{OR} + \overrightarrow{RS} + \overrightarrow{SQ}. \tag{1}$$

The first vector can be written in the following way:

$$\overrightarrow{OR} = \mathbf{n}|OR| = \mathbf{n}(\mathbf{n}\cdot\mathbf{x}),$$

where $\cdot$ denotes the operator of the classical dot product. The second vector which occurs in Eqn. (1) may be presented as

$$\begin{aligned}\overrightarrow{RS} &= \frac{\overrightarrow{RP}}{|RP|}|RS| \\ &= \frac{\overrightarrow{OP}-\overrightarrow{OR}}{r} r\cos\varphi \\ &= [\mathbf{x}-\mathbf{n}(\mathbf{n}\cdot\mathbf{x})]\cos\varphi.\end{aligned}$$

The last component in Eqn. (1) is perpendicular to the plane defined by the vectors $\overrightarrow{OR}$ and $\overrightarrow{OP}$, so its direction is defined by the vector $\mathbf{n}\times\mathbf{x}$, where $\times$ denotes the operator of the classical cross product. Its length is $|SQ| = |RQ|\sin\varphi = r\sin\varphi$. Hence, we can write

$$\overrightarrow{SQ} = \frac{\mathbf{n}\times\mathbf{x}}{|\mathbf{n}\times\mathbf{x}|} r\sin\varphi = (\mathbf{n}\times\mathbf{x})\sin\varphi.$$

Using the above relationships, after some basic transformation, Eqn. (1) may be presented in the following way:

$$\mathbf{y} = \mathbf{x}\cos\varphi + \mathbf{n}(\mathbf{n}\cdot\mathbf{x})(1-\cos\varphi) + (\mathbf{n}\times\mathbf{x})\sin\varphi, \tag{2}$$

which is well known as the rotation formula (Goldstein, 1980) or Rodrigues' rotation formula.

If we chose a Cartesian coordinate system with the origin at the point O and a standard base, then Eqn. (2) would be as follows:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \tag{3}$$

where $\mathbf{y} = [y_1, y_2, y_3]^{\mathrm{T}}$, $\mathbf{x} = [x_1, x_2, x_3]^{\mathrm{T}}$,

$$\begin{aligned}
r_{11} &= \cos\varphi + n_1^2(1-\cos\varphi),\\
r_{12} &= n_1 n_2(1-\cos\varphi) - n_3\sin\varphi,\\
r_{13} &= n_1 n_3(1-\cos\varphi) + n_2\sin\varphi,\\
r_{21} &= n_1 n_2(1-\cos\varphi) + n_3\sin\varphi,\\
r_{22} &= \cos\varphi + n_2^2(1-\cos\varphi),\\
r_{23} &= n_2 n_3(1-\cos\varphi) - n_1\sin\varphi,\\
r_{31} &= n_1 n_3(1-\cos\varphi) - n_2\sin\varphi,\\
r_{32} &= n_2 n_3(1-\cos\varphi) + n_1\sin\varphi,\\
r_{33} &= \cos\varphi + n_3^2(1-\cos\varphi),
\end{aligned}$$

and $\mathbf{n} = [n_1, n_2, n_3]^{\mathrm{T}}$.

The matrix which occurs in (3) is known as the rotation matrix. This provides an algebraic description of rotations, and is used extensively for computations in geometry, physics, and computer graphics.

A general rotation can be composed out of particular rotations along some specified (coordinate) axes, as is done, for example, in Euler-angle parametrization.

3. Quaternions and their connection with rotations

A quaternion is a hypercomplex number which can be presented as a four-element vector or as a sum (Kantor and Solodovnikov, 1989):

$$q = \begin{bmatrix} q_1\\ q_2\\ q_3\\ q_4 \end{bmatrix} = q_1 i + q_2 j + q_3 k + q_4,$$

where $\{q_l\}$, $l = 1,\dots,4$, are real numbers, while i, j, k are imaginary units whose products are defined by the following equations:

$$\begin{aligned}
&ij = k, \quad && jk = i, \quad && ki = j, \quad && ji = -k,\\
&kj = -i, && ik = -j, && ijk = -1.
\end{aligned}$$

Addition and multiplication of quaternions are defined exactly analogously to complex addition and multiplication, but it must be remembered that multiplication is not commutative. A quaternion has a real part q_4 and an imaginary part $q_1 i + q_2 j + q_3 k$. The latter has three components and, thus, can be used as a vector quantity. For this reason, the real part is sometimes referred to as the scalar part of the quaternion and the whole quaternion may be represented by the sum of its vector and scalar parts. A quaternion with a zero real or scalar part is called a pure quaternion. Since $q_1 i + q_2 j + q_3 k$ is a three-dimensional vector, clearly there is a one-to-one correspondence between vectors in 3D space and the quaternion sub-space consisting of pure quaternions. The modulus (norm) and conjugate of a quaternion follow the definitions for complex numbers,

$$\begin{aligned}
|q| &= \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2},\\
q^* &= -q_1 i - q_2 j - q_3 k + q_4.
\end{aligned}$$

A quaternion with a unit modulus is called a unit quaternion. The conjugate of a quaternion is obtained, like the complex conjugate, by negating the imaginary or vector part. For every non-zero quaternion, an inverse quaternion is given by

$$q^{-1} = \frac{q^*}{|q|^2}.$$

Similarly to complex numbers, the quaternion has also trigonometric form,

$$q = |q|(\mathbf{n}\sin\alpha + \cos\alpha),$$

where $\mathbf{n} = n_1 i + n_2 j + n_3 k$ is a unit pure quaternion. From this representation it follows that every quaternion is connected with a unit vector $\mathbf{n}$, which may be treated as a vector in 3D space and an angle, so it has the same parameters as rotations.

Quaternions are widely used to describe rotations in 3D space. Compared with rotation matrices, they are more compact, more numerically stable, and may be more efficient. Compared with Euler angles, they are simpler to compose and avoid the problem of gimbal lock.

We can interpret the rotation of the vector $\mathbf{x} = [x_1, x_2, x_3]^{\mathrm{T}}$ by the angle φ about the axis represented by the unit vector $\mathbf{n} = [n_1, n_2, n_3]^{\mathrm{T}}$ using the quaternion product as follows (Mukundan, 2002; Markley, 2008; Shuster and Natanson, 1993):

$$y = qxq^{-1}, \tag{4}$$

where $q = in_1\sin\frac{\varphi}{2} + jn_2\sin\frac{\varphi}{2} + kn_3\sin\frac{\varphi}{2} + \cos\frac{\varphi}{2}$ is a unit quaternion, while $x = x_1 i + x_2 j + x_3 k$ and $y = y_1 i + y_2 j + y_3 k$ are quaternion counterparts of the vectors $\mathbf{x}$ and $\mathbf{y}$. Since the quaternion q is a unit quaternion, the inverse $q^{-1} = q^*$. A full justification of (4) can be found in many online sources (e.g., Jia, 2015; Vicci, 2001).

If we calculate the product of quaternions $q = q_1 i + q_2 j + q_3 k + q_4$, $x = x_1 i + x_2 j + x_3 k$ and $q^* = -q_1 i - q_2 j - q_3 k + q_4$, according to the formula (4), without taking into account the relation between the quaternions q and q^*, we must perform 12 multiplications and 8 additions of real numbers when determining the product qx (since x is determined by only 3 real numbers), and 12 multiplications and 9 additions of real numbers when determining the product $(qx)q*$ (since y has only 3 real components). The total number of multiplications and additions of real numbers in the calculation of quaternion y will be equal to 24 and 17, respectively.

If we take into account the relation between quaternions q and q^*, and do the quaternion multiplication according to Eqn. (4), then we obtain

$$\begin{aligned} y &= y_1 i + y_2 j + y_3 k \\ &= \big[(q_1^2 - q_2^2 - q_3^2 + q_4^2)x_1 \\ &\quad + 2(q_1q_2 - q_3q_4)x_2 \\ &\quad + 2(q_1q_3 + q_2q_4)x_3\big] i \\ &\quad + \big[2(q_1q_2 + q_3q_4)x_1 \\ &\quad + (-q_1^2 + q_2^2 - q_3^2 + q_4^2)x_2 \\ &\quad + 2(-q_1q_4 + q_2q_3)x_3\big] j \\ &\quad + \big[2(q_1q_3 - q_2q_4)x_1 \\ &\quad + 2(q_1q_4 + q_2q_3)x_2 \\ &\quad + (-q_1^2 - q_2^2 + q_3^2 + q_4^2)x_3\big] k. \end{aligned} \tag{5}$$

While we calculate the quaternion y, 6 conventional multiplications, 4 operations of squaring, 6 trivial multiplications by 2 (shifts), and 15 additions are required to calculate the expressions in parentheses. Then multiplying these results by the components x_1, x_2 and x_3 of quaternion x require further 9 multiplications, and in the end obtaining the sums in square brackets require another 6 additions. Thus, 15 multiplications, 4 squaring operations, 6 multiplications by 2 (which we do not count) and 21 additions are required to rotate the vector in 3D space. Then the implementation of such computations requires $15 + 4 = 19$ multipliers.

From now on we will write subscripts to emphasise the sizes of vectors and matrices. Using the matrix-vector notation, Eqn. (5) can be written as follows:

$$\mathbf{y}_4 = \mathbf{Q}_4 \mathbf{Q}_4^* \mathbf{x}_4, \tag{6}$$

where

$$\mathbf{Q}_4 = \begin{bmatrix} q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{bmatrix},$$

$$\mathbf{Q}_4^* = \begin{bmatrix} q_4 & -q_3 & q_2 & -q_1 \\ q_3 & q_4 & -q_1 & -q_2 \\ -q_2 & q_1 & q_4 & -q_3 \\ q_1 & q_2 & q_3 & q_4 \end{bmatrix},$$

$\mathbf{x}_4 = [x_1, x_2, x_3, 0]^{\mathrm{T}}$ and $\mathbf{y}_4 = [y_1, y_2, y_3, 0]^{\mathrm{T}}$ are input and output vectors, respectively. In this context, $\mathbf{Q}_4$ is a well-known matrix representation of the quaternion q and "conjugation" of a matrix, marked with an asterisk, is not a matrix representation of a conjugate quaternion but means the change of the signs of the fourth row and the fourth column entries of this matrix (the entry which is in the fourth row and in the fourth column will change the sign twice, i.e., it will not change).

To show that Eqn. (5) can be written equivalently as (6), we calculate the product of the matrices $\mathbf{Q}_4$ and $\mathbf{Q}_4^*$ in the expression (6),

$$\begin{aligned} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ 0 \end{bmatrix} &= \begin{bmatrix} q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{bmatrix} \\ &\quad \cdot \begin{bmatrix} q_4 & -q_3 & q_2 & -q_1 \\ q_3 & q_4 & -q_1 & -q_2 \\ -q_2 & q_1 & q_4 & -q_3 \\ q_1 & q_2 & q_3 & q_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 0 \end{bmatrix} \\ &= \left[\begin{array}{ccc:c} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ \hdashline 0 & 0 & 0 & r_{44} \end{array}\right] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 0 \end{bmatrix}, \end{aligned}$$

where

$$\begin{aligned} r_{11} &= q_1^2 - q_2^2 - q_3^2 + q_4^2, \\ r_{12} &= 2(q_1q_2 - q_3q_4), \\ r_{13} &= 2(q_1q_3 + q_2q_4), \\ r_{21} &= 2(q_1q_2 + q_3q_4), \\ r_{22} &= -q_1^2 + q_2^2 - q_3^2 + q_4^2, \\ r_{23} &= 2(q_2q_3 - q_1q_4), \\ r_{31} &= 2(q_1q_3 - q_2q_4), \\ r_{32} &= 2(q_2q_3 + q_1q_4), \\ r_{33} &= -q_1^2 - q_2^2 + q_3^2 + q_4^2, \\ r_{44} &= q_1^2 + q_2^2 + q_3^2 + q_4^2. \end{aligned}$$

Then we can write

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}. \tag{7}$$

It is an exact equivalent of Eqn. (5) in vector-matrix notation.

It can be directly checked that the matrix in the expression (7) is equal to the rotation matrix from Eqn. (2) when we use the relations $q_1 = n_1 \sin\frac{\varphi}{2}$, $q_2 = n_2 \sin\frac{\varphi}{2}$, $q_3 = n_3 \sin\frac{\varphi}{2}$, $q_4 = \cos\frac{\varphi}{2}$. That is why we use the same symbols for the entries of the rotation matrix.

It is easy to see that the calculation of the entries of the rotation matrix, which occurs in (7), requires 6 conventional multiplications, 4 operations of squaring, 6 trivial multiplications by 2 (shifts), and 15 additions. Multiplication of a vector by this matrix requires 9 more multiplications and 6 additions. Thus, 15 multiplications, 4 squaring operations, 6 multiplications by 2, and 21 additions are required to rotate the vector in space. It should be noted that squares are a special case of multiplication where both operands are identical. For this reason, digital circuits designers often use embedded general-purpose multipliers to implement the squaring units by connecting a multiplier's inputs together. Then a completely parallel hardware implementation of such computations requires $15 + 4 = 19$ multipliers. The multiplier is the most critical unit, so reducing the number of multipliers plays a significant role. We will show how this can be done.

Since the starting point of the described algorithm of rotation in 3D space is Eqn. (6), we will discuss how this description can be applied to the sequence of rotations. It is well known that a vector connected to a pure quaternion x undergoing two rotations, represented by the unit quaternions p and q, respectively, will have the same outcome as undergoing a single rotation. This rotation is related to the quaternion qp, because the result of these rotations is equal to

$$q(pxp^*)q^* = (qp)x(p^*q^*) = (qp)x(qp)^*. \tag{8}$$

In this equation, we used the facts that quaternion multiplication is associative and has the following property:

$$(qp)^* = p^*q^*. \tag{9}$$

Thus, if we want to find the result of many rotations, it is enough to calculate the product of quaternions corresponding to these rotations, but in reverse order.

In the matrix-vector notation of rotation in 3D space, as in Eqn. (6), if the quaternions $p = p_1i+p_2j+p_3k+p_4$ and $q = q_1i + q_2j + q_3k + q_4$ correspond to the matrices $\mathbf{P}_4$ and $\mathbf{Q}_4$, respectively, where

$$\mathbf{P}_4 = \begin{bmatrix} p_4 & -p_3 & p_2 & p_1 \\ p_3 & p_4 & -p_1 & p_2 \\ -p_2 & p_1 & p_4 & p_3 \\ -p_1 & -p_2 & -p_3 & p_4 \end{bmatrix},$$

$$\mathbf{Q}_4 = \begin{bmatrix} q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{bmatrix},$$

then, according to (6), the result of composition of rotation will be described by the vector

$$\mathbf{Q}_4\mathbf{Q}_4^*(\mathbf{P}_4\mathbf{P}_4^*\mathbf{x}_4). \tag{10}$$

It is easy to directly check that

$$\mathbf{Q}_4^*\mathbf{P}_4 = \mathbf{P}_4\mathbf{Q}_4^* \tag{11}$$

and, unlike the case of the quaternions,

$$(\mathbf{Q}_4\mathbf{P}_4)^* = \mathbf{Q}_4^*\mathbf{P}_4^*. \tag{12}$$

These two properties allow us to represent the expression (10) in the following form:

$$(\mathbf{Q}_4\mathbf{P}_4)(\mathbf{Q}_4\mathbf{P}_4)^*\mathbf{x}_4. \tag{13}$$

It should be noted that the product $\mathbf{Q}_4\mathbf{P}_4$ is the matrix equivalent of the quaternion product qp. Thus, similarly as in the case of quaternion notation, to find the result of many rotations, it is enough to calculate the product of matrices being the matrix representation of the corresponding quaternions, but in reverse order. It might seem that calculating the product of two $4{\times}4$ matrices requires more operations on real numbers than calculating the product of two quaternions. However, among the 16 entries in each such matrix, only 4 entries differ (e.g., entries of a selected row or column), and the remaining entries repeat these values. Therefore, to perform a set of consecutive rotations using matrix-vector products, the same number of operations with real numbers is required as in the case of using conventional quaternion multiplication.

4. Synthesis of a rationalized algorithm for quaternion-based 3D rotation

The idea is that both matrices $\mathbf{Q}_4$ and $\mathbf{Q}_4^*$ in the expression (6) can be decomposed as an algebraic sum of a symmetric Toeplitz matrix and another matrix which has many zero entries. The Toeplitz matrix is shift-structured, and a number of algorithms exist for fast matrix-vector multiplication. For instance, the matrix can be diagonalized using the fast Hadamard transform (FHT), and thus matrix-vector products can be computed efficiently.

Let us multiply the last column of $\mathbf{Q}_4^*$ by (-1). The effect of this multiplication is equivalent to changing the last coordinate of the vector through which the matrix $\mathbf{Q}_4^*$ is multiplied to the opposite. This treatment is undertaken so that the obtained matrix can be written as an algebraic sum of the block-symmetric Toeplitz-type matrix and some sparse matrix, i.e., a matrix containing only a small number of nonzero entries. We can easily see that such transformation leads to minimization of the computational complexity of the final algorithm. We will denote by $\mathbf{z}_4 = [z_1, z_2, z_3, z_4]^{\mathrm{T}}$ the product $\mathbf{Q}_4^*\mathbf{x}_4$, which occurs in Eqn. (6). Then we can rewrite $\mathbf{z}_4$ in the following form:

$$\mathbf{z}_4 = \mathbf{Q}_4^*\mathbf{x}_4 = (\hat{\mathbf{Q}}_4 - 2\check{\mathbf{Q}}_4)\bar{\mathbf{x}}_4, \tag{14}$$

where

$$\hat{\mathbf{Q}}_\mathbf{4} = \left[\begin{array}{cc:cc} q_4 & q_3 & q_2 & q_1 \\ q_3 & q_4 & q_1 & q_2 \\ \hdashline q_2 & q_1 & q_4 & q_3 \\ q_1 & q_2 & q_3 & q_4 \end{array}\right],$$

$$\check{\mathbf{Q}}_4 = \left[\begin{array}{cc:cc} 0 & q_3 & 0 & 0 \\ 0 & 0 & q_1 & 0 \\ \hdashline q_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_4 \end{array}\right],$$

and $\bar{\mathbf{x}}_4 = [x_1, x_2, x_3, -x_4]^{\mathrm{T}}$. In this case the matrix-vector product $\hat{\mathbf{Q}}_4\bar{\mathbf{x}}_4$ (with the Toeplitz-type matrix) can now be calculated using one of the well-known fast algorithms. Indeed, it is not hard to see that the matrix has the following structure:

$$\hat{\mathbf{Q}}_4 = \left[\begin{array}{c|c} \mathbf{A}_2 & \mathbf{B}_2 \\ \hline \mathbf{B}_2 & \mathbf{A}_2 \end{array}\right],$$

where

$$\mathbf{A}_2 = \begin{bmatrix} q_4 & q_3 \\ q_3 & q_4 \end{bmatrix}, \quad \mathbf{B}_2 = \begin{bmatrix} q_2 & q_1 \\ q_1 & q_2 \end{bmatrix}.$$

The matrix possessing such a structure can be effectively factorized (Cariow, 2014), and we can calculate the product of $\hat{\mathbf{Q}}_4\bar{\mathbf{x}}_4$ with the following decomposition:

$$\hat{\mathbf{Q}}_4\bar{\mathbf{x}}_4 = \Big\{ (\mathbf{H}_2 \otimes \mathbf{I}_2) \frac{1}{2} [(\mathbf{A}_2 + \mathbf{B}_2) \oplus (\mathbf{A}_2 - \mathbf{B}_2)] (\mathbf{H}_2 \otimes \mathbf{I}_2) \Big\} \bar{\mathbf{x}}_4, \tag{15}$$

where

$$\mathbf{A}_2 + \mathbf{B}_2 = \left[\begin{array}{c|c} q_4 + q_2 & q_3 + q_1 \\ \hline q_3 + q_1 & q_4 + q_2 \end{array}\right],$$

$$\mathbf{A}_2 - \mathbf{B}_2 = \left[\begin{array}{c|c} q_4 - q_2 & q_3 - q_1 \\ \hline q_3 - q_1 & q_4 - q_2 \end{array}\right],$$

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

is the second-order Hadamard matrix, $\mathbf{I}_N$ is the identity matrix of order N, and "$\otimes$" and "$\oplus$" denote the Kronecker product and the direct sum of two matrices, respectively (Steeb and Hardy, 2011).

It is easy to see that the matrices $\mathbf{A}_2$+$\mathbf{B}_2$ and $\mathbf{A}_2$–$\mathbf{B}_2$ possess a structure that provides "good" factorization, too. Therefore, we can write as previously:

$$\mathbf{A}_2 + \mathbf{B}_2 = \mathbf{H}_2 \Big\{ \frac{1}{4} \Big[\big(q_4 + q_2 + q_3 + q_1\big) \oplus \big(q_4 + q_2 - q_3 - q_1\big) \Big] \Big\} \mathbf{H}_2,$$

$$\mathbf{A}_2 - \mathbf{B}_2 = \mathbf{H}_2 \Big\{ \frac{1}{4} \Big[\big(q_4 - q_2 + q_3 - q_1\big) \oplus \big(q_4 - q_2 - q_3 + q_1\big) \Big] \Big\} \mathbf{H}_2.$$

Combining partial decompositions in a single procedure, we can rewrite (15) as follows:

$$\hat{\mathbf{Q}}_4\bar{\mathbf{x}}_4 = \mathbf{W}_4^{(0)} \mathbf{W}_4^{(1)} \mathbf{D}_4 \mathbf{W}_4^{(1)} \mathbf{W}_4^{(0)} \hat{\mathbf{I}}_4 \mathbf{x}_4, \tag{16}$$

where

$$\mathbf{W}_4^{(0)} = \mathbf{H}_2 \otimes \mathbf{I}_2 = \left[\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ \hline 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{array}\right],$$

$$\mathbf{W}_4^{(1)} = \mathbf{I}_2 \otimes \mathbf{H}_2 = \left[\begin{array}{cc|cc} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{array}\right],$$

$$\mathbf{D}_4 = \operatorname{diag}(s_1, s_2, s_3, s_4),$$

$$\hat{\mathbf{I}}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

The diagonal entries of the matrix $\mathbf{D}_4$ are as follows:

$$s_1 = \frac{1}{4}(q_4 + q_2 + q_3 + q_1),$$
$$s_2 = \frac{1}{2}(q_4 + q_2 - q_3 - q_1),$$
$$s_3 = \frac{1}{4}(q_4 - q_2 + q_3 - q_1),$$
$$s_4 = \frac{1}{2}(q_4 - q_2 - q_3 + q_1).$$

It is easy to check that the entries of the matrix $\mathbf{D}_4$ can be calculated using the following matrix-vector procedure, which reduces the number of underlying real additions:

$$\mathbf{d}_4 = \frac{1}{4} \mathbf{W}_4^{(1)} \mathbf{W}_4^{(0)} \mathbf{J}_4 \mathbf{q}_4, \tag{17}$$

where

$$\mathbf{d}_4 = [s_1, s_2, s_3, s_4]^{\mathrm{T}},$$

$$\mathbf{J}_4 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

is the exchange matrix, and

$$\mathbf{q}_4 = [q_1, q_2, q_3, q_4]^{\mathrm{T}}.$$

Unfortunately, the computational complexity of the product $2\check{\mathbf{Q}}_4\bar{\mathbf{x}}_4$, which occurs in Eqn. (14), cannot be reduced and this product is calculated directly, without any tricks. Combining the calculations for both products $\hat{\mathbf{Q}}_4\bar{\mathbf{x}}_4$ and $2\check{\mathbf{Q}}_4\bar{\mathbf{x}}_4$ in a single procedure, we finally obtain

$$\begin{aligned} \mathbf{z}_4 &= \mathbf{Q}_4^* \mathbf{x}_4 \\ &= \mathbf{\Sigma}_{4\times 8} \mathbf{W}_8^{(0)} \mathbf{W}_8^{(1)} \mathbf{D}_8 \mathbf{W}_8^{(1)} \check{\mathbf{W}}_8^{(0)} \mathbf{P}_{8\times 4}^{(0)} \mathbf{x}_4, \end{aligned} \tag{18}$$

where

$$\begin{aligned}
\mathbf{\Sigma}_{4\times 8} &= (\bar{\mathbf{1}}_{1\times 2}\otimes \mathbf{I}_4)\\
&= \left[\begin{array}{cccc:cccc}
1 & 0 & 0 & 0 & -1 & 0 & 0 & 0\\
0 & 1 & 0 & 0 & 0 & -1 & 0 & 0\\
0 & 0 & 1 & 0 & 0 & 0 & -1 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & 0 & -1
\end{array}\right],\\
\bar{\mathbf{1}}_{1\times 2} &= [1,-1],\\
\mathbf{W}_8^{(0)} &= (\mathbf{H}_2\otimes \mathbf{I}_2)\oplus \mathbf{I}_4\\
&= \left[\begin{array}{c:c}
\begin{array}{cc:cc}
1 & 0 & 1 & 0\\
0 & 1 & 0 & 1\\ \hdashline
1 & 0 & -1 & 0\\
0 & 1 & 0 & -1
\end{array} & \mathbf{0}_4\\ \hdashline
\mathbf{0}_4 & \begin{array}{cccc}
1 & 0 & 0 & 0\\
0 & 1 & 0 & 0\\
0 & 0 & 1 & 0\\
0 & 0 & 0 & 1
\end{array}
\end{array}\right],\\
\mathbf{W}_8^{(1)} &= (\mathbf{I}_2\otimes \mathbf{H}_2)\oplus \mathbf{I}_4\\
&= \left[\begin{array}{c:c}
\begin{array}{c:c}
\begin{array}{cc} 1 & 1\\ 1 & -1\end{array} & \mathbf{0}_2\\ \hdashline
\mathbf{0}_2 & \begin{array}{cc} 1 & 1\\ 1 & -1\end{array}
\end{array} & \mathbf{0}_4\\ \hdashline
\mathbf{0}_4 & \begin{array}{cccc}
1 & 0 & 0 & 0\\
0 & 1 & 0 & 0\\
0 & 0 & 1 & 0\\
0 & 0 & 0 & 1
\end{array}
\end{array}\right],
\end{aligned}$$

$$\mathbf{D}_8 = \mathrm{diag}(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8),$$

$$s_5 = 2q_3,\quad s_6 = 2q_1,\quad s_7 = 2q_2,\quad s_8 = 2q_4,$$

and s_1, s_2, s_3, s_4 are the diagonal entries of matrix $\mathbf{D}_4$,

$$\begin{aligned}
\check{\mathbf{W}}_8^{(0)} &= (\mathbf{H}_2\otimes \mathbf{I}_2)\oplus \mathbf{P}_4^{(0)}\\
&= \left[\begin{array}{c:c}
\begin{array}{cc:cc}
1 & 0 & 1 & 0\\
0 & 1 & 0 & 1\\ \hdashline
1 & 0 & -1 & 0\\
0 & 1 & 0 & -1
\end{array} & \mathbf{0}_4\\ \hdashline
\mathbf{0}_4 & \begin{array}{cccc}
0 & 1 & 0 & 0\\
0 & 0 & 1 & 0\\
1 & 0 & 0 & 0\\
0 & 0 & 0 & 1
\end{array}
\end{array}\right],\\
\mathbf{P}_4^{(0)} &= \begin{bmatrix}
0 & 1 & 0 & 0\\
0 & 0 & 1 & 0\\
1 & 0 & 0 & 0\\
0 & 0 & 0 & 1
\end{bmatrix},
\end{aligned}$$

$$\begin{aligned}
\mathbf{P}_{8\times 4}^{(0)} = \mathbf{1}_{2\times 1}\otimes \hat{\mathbf{I}}_4 &= \left[\begin{array}{cccc}
1 & 0 & 0 & 0\\
0 & 1 & 0 & 0\\
0 & 0 & 1 & 0\\
0 & 0 & 0 & -1\\ \hdashline
1 & 0 & 0 & 0\\
0 & 1 & 0 & 0\\
0 & 0 & 1 & 0\\
0 & 0 & 0 & -1
\end{array}\right],\\
\mathbf{1}_{2\times 1} &= [1,1]^{\mathrm{T}}.
\end{aligned}$$

It is not hard to see that the entries of the matrix $\mathbf{D}_8$ can be calculated using the following matrix-vector procedure:

$$\mathbf{d}_8 = \hat{\mathbf{D}}_8^{(0)}\mathbf{W}_8^{(1)}\tilde{\mathbf{W}}_8^{(0)}\mathbf{P}_{8\times 4}^{(1)}\mathbf{J}_4\mathbf{q}_4, \tag{19}$$

where

$$\begin{aligned}
\mathbf{d}_8 &= [s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8]^{\mathrm{T}},\\
\hat{\mathbf{D}}_8^{(0)} &= \mathrm{diag}\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 2, 2, 2, 2\right),\\
\tilde{\mathbf{W}}_8^{(0)} &= (\mathbf{H}_2\otimes \mathbf{I}_2)\oplus \mathbf{P}_4^{(1)}\\
&= \left[\begin{array}{c:c}
\begin{array}{cc:cc}
1 & 0 & 1 & 0\\
0 & 1 & 0 & 1\\ \hdashline
1 & 0 & -1 & 0\\
0 & 1 & 0 & -1
\end{array} & \mathbf{0}_4\\ \hdashline
\mathbf{0}_4 & \begin{array}{cccc}
0 & 1 & 0 & 0\\
0 & 0 & 0 & 1\\
0 & 0 & 1 & 0\\
1 & 0 & 0 & 0
\end{array}
\end{array}\right],
\end{aligned}$$

$$\mathbf{P}_4^{(1)} = \begin{bmatrix}
0 & 1 & 0 & 0\\
0 & 0 & 0 & 1\\
0 & 0 & 1 & 0\\
1 & 0 & 0 & 0
\end{bmatrix},$$

$$\mathbf{P}_{8\times 4}^{(1)} = \mathbf{1}_{2\times 1}\otimes \mathbf{I}_4 = \left[\begin{array}{cccc}
1 & 0 & 0 & 0\\
0 & 1 & 0 & 0\\
0 & 0 & 1 & 0\\
0 & 0 & 0 & 1\\ \hdashline
1 & 0 & 0 & 0\\
0 & 1 & 0 & 0\\
0 & 0 & 1 & 0\\
0 & 0 & 0 & 1
\end{array}\right].$$

Now we will go back to Eqn. (6). First, we have shown how to calculate the product $\mathbf{Q}_4^{*}\mathbf{x}_4 = \mathbf{z}_4$. Subsequently, we will focus on calculation of the product $\mathbf{Q}_4\mathbf{z}_4$. We multiply the last row of $\mathbf{Q}_4$ by (-1). Then it is not hard to see that the matrix-vector product $\mathbf{Q}_4\mathbf{z}_4$ can be represented in the following form:

$$\mathbf{y}_4 = \mathbf{Q}_4\mathbf{z}_4 = \hat{\mathbf{I}}_4(\hat{\mathbf{Q}}_4 - 2\check{\mathbf{Q}}_4)\mathbf{z}_4. \tag{20}$$

It is easy to see that we are again dealing with the same matrices. Thus, in this case all the previous steps related to the synthesis of the algorithm for calculating the product $\mathbf{z}_4 = \mathbf{Q}_4^* \mathbf{x}_4$ can be repeated for calculating the product $\mathbf{Q}_4\mathbf{z}_4$, too. Thus, we can describe the final procedure for computing the vector $\mathbf{y}_4$ as follows:

$$\begin{aligned}\mathbf{y}_4 = \hat{\mathbf{I}}_4 \mathbf{\Sigma}_{4\times 8} \mathbf{W}_8^{(0)} \mathbf{W}_8^{(1)} \mathbf{D}_8 \mathbf{W}_8^{(1)} \check{\mathbf{W}}_8^{(0)} \mathbf{P}_{8\times 4}^{(1)} \\ \cdot \mathbf{\Sigma}_{4\times 8} \mathbf{W}_8^{(0)} \mathbf{W}_8^{(1)} \mathbf{D}_8 \mathbf{W}_8^{(1)} \check{\mathbf{W}}_8^{(0)} \mathbf{P}_{8\times 4}^{(0)} \mathbf{x}_4. \quad (21)\end{aligned}$$

Realization of computations via (21) requires 16 multiplications and 48 additions of real numbers. However, taking into account that $x_4 = 0$ and $y_4 = 0$, the number of operations can be further reduced. Then the expression (21) can be rewritten in the following refined form:

$$\begin{aligned}\mathbf{y}_3 = \mathbf{\Sigma}_{3\times 7} \bar{\mathbf{W}}_7^{(0)} \mathbf{W}_7^{(1)} \mathbf{D}_7 \mathbf{W}_7^{(1)} \hat{\mathbf{W}}_7^{(0)} \mathbf{P}_{7\times 4}^{(1)} \\ \cdot \mathbf{\Sigma}_{4\times 7} \mathbf{W}_7^{(0)} \mathbf{W}_7^{(1)} \mathbf{D}_7 \mathbf{W}_7^{(1)} \check{\mathbf{W}}_7^{(0)} \mathbf{P}_{7\times 3}^{(0)} \mathbf{x}_3, \quad (22)\end{aligned}$$

where

$$\mathbf{\Sigma}_{3\times 7} = \left[\begin{array}{ccc|c|ccc} 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{array}\right],$$

$$\bar{\mathbf{W}}_7^{(0)} = \left[\begin{array}{cccc|ccc} 1 & 0 & 1 & 0 & & & \\ 0 & 1 & 0 & 0 & & \mathbf{0}_{4\times 3} & \\ 1 & 0 & -1 & 0 & & & \\ 0 & 0 & 0 & 1 & & & \\ \hline & & & & 1 & 0 & 0 \\ & \mathbf{0}_{3\times 4} & & & 0 & 1 & 0 \\ & & & & 0 & 0 & 1 \end{array}\right],$$

$$\mathbf{W}_7^{(1)} = (\mathbf{I}_2 \otimes \mathbf{H}_2) \oplus \mathbf{I}_3 = \left[\begin{array}{cc|cc|ccc} 1 & 1 & & & & & \\ 1 & -1 & \multicolumn{2}{c|}{\mathbf{0}_2} & & \mathbf{0}_{4\times 3} & \\ \cline{1-4} & & 1 & 1 & & & \\ \multicolumn{2}{c|}{\mathbf{0}_2} & 1 & -1 & & & \\ \hline & & & & 1 & 0 & 0 \\ & \mathbf{0}_{3\times 4} & & & 0 & 1 & 0 \\ & & & & 0 & 0 & 1 \end{array}\right],$$

$$\mathbf{D}_7 = \operatorname{diag}(s_1, s_2, s_3, s_4, s_5, s_6, s_7),$$

$$\hat{\mathbf{W}}_7^{(0)} = \left[\begin{array}{cccc|ccc} 1 & 0 & 1 & 0 & & & \\ 0 & 1 & 0 & 0 & & \mathbf{0}_{4\times 3} & \\ 1 & 0 & -1 & 0 & & & \\ 0 & 0 & 0 & 1 & & & \\ \hline & & & & 0 & 1 & 0 \\ & \mathbf{0}_{3\times 4} & & & 0 & 0 & 1 \\ & & & & 1 & 0 & 0 \end{array}\right],$$

$$\mathbf{P}_{7\times 4}^{(1)} = \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right],$$

$$\mathbf{\Sigma}_{4\times 7} = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right],$$

$$\mathbf{W}_7^{(0)} = (\mathbf{H}_2 \otimes \mathbf{I}_2) \oplus \mathbf{I}_3 = \left[\begin{array}{cc|cc|ccc} 1 & 0 & 1 & 0 & & & \\ 0 & 1 & 0 & 1 & & \mathbf{0}_{4\times 3} & \\ \cline{1-4} 1 & 0 & -1 & 0 & & & \\ 0 & 1 & 0 & -1 & & & \\ \hline & & & & 1 & 0 & 0 \\ & \mathbf{0}_{3\times 4} & & & 0 & 1 & 0 \\ & & & & 0 & 0 & 1 \end{array}\right],$$

$$\check{\mathbf{W}}_7^{(0)} = \left[\begin{array}{cccc|ccc} 1 & 0 & 1 & 0 & & & \\ 0 & 1 & 0 & 0 & & \mathbf{0}_{4\times 3} & \\ 1 & 0 & -1 & 0 & & & \\ 0 & 0 & 0 & 1 & & & \\ \hline & & & & 0 & 1 & 0 \\ & \mathbf{0}_{3\times 4} & & & 0 & 0 & 1 \\ & & & & 1 & 0 & 0 \end{array}\right],$$

$$\mathbf{P}_{7\times 3}^{(0)} = \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \hline 0 & 1 & 0 \\ \hline 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right].$$

In turn, the entries of the matrix $\mathbf{D}_7 = \operatorname{diag}(\mathbf{d}_7)$ can be calculated using the following improved matrix-vector procedure:

$$\mathbf{d}_7 = \mathbf{D}_7^{(1)} \mathbf{W}_7^{(1)} \mathbf{W}_7^{(2)} \mathbf{P}_{7\times 4}^{(2)} \mathbf{J}_4 \mathbf{q}_4, \quad (23)$$

where

$$\mathbf{d}_7 = [s_1, s_2, s_3, s_4, s_5, s_6, s_7]^{\mathrm{T}},$$

$$\mathbf{D}_7^{(1)} = \operatorname{diag}\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 2, 2, 2\right),$$

$$\mathbf{W}_7^{(2)} = \left[\begin{array}{cc|cc|ccc} 1 & 0 & 1 & 0 & & & \\ 0 & 1 & 0 & 1 & & \mathbf{0}_{4\times 3} & \\ \cline{1-4} 1 & 0 & -1 & 0 & & & \\ 0 & 1 & 0 & -1 & & & \\ \hline & & & & 1 & 0 & 0 \\ & \mathbf{0}_{3\times 4} & & & 0 & 0 & 1 \\ & & & & 0 & 1 & 0 \end{array}\right],$$

$$\mathbf{P}_{7\times4}^{(2)} = \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hdashline 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right].$$

Figure 2 shows a data flow diagram of the rationalized algorithm for quaternion-based 3D rotation and Fig. 3 shows a data flow diagram of the process for calculating entries of the matrix $\mathbf{D}_7$ in accordance with the procedure (23). In this paper, data flow diagrams are oriented from left to right. Straight lines in the figures denote the operations of data transfer. Points where lines converge denote summation. The dotted lines indicate the subtraction operation. We use the usual lines without arrows intentionally, so as not to clutter the picture. The circles in these figures show the operation of multiplication by a number (variable or constant) inscribed inside a circle. In turn, the rectangles indicate the matrix-vector multiplications with the Hadamard matrices of order 2.

5. Performance evaluation

Let us now consider the data flow diagrams, presented in Figs. 2 and 3. One can observe that realization of computations in accordance with (22) and (23) requires only 14 multiplications, 43 additions, 4 right-shifts (multiplications by 1/4), and 3 left-shifts (multiplications by 2).

Table 1 presents a comparison of the number of multiplications and additions of real numbers, which are necessary for calculating the rotated vector in 3D space, using the quaternion multiplication, the rotation matrix and the proposed algorithm.

As we can see, the implementation of our algorithm requires fewer multiplications than the implementation of calculations based on the rotation matrix-vector product. Reducing the number of multiplications is especially important in the design of specialized VLSI fully parallel processors because minimizing the number of necessary multipliers also reduces the power dissipation and lowers the cost implementation of the entire system being implemented. This is because a hardware multiplier is more complicated unit than an adder and occupies much more of the chip area than the adder. It is proved that the hardware complexity of an embedded multiplier grows quadratically with the operand size, while the hardware complexity of a binary adder increases linearly with the operand size. Thus, we want to note that our algorithm is mainly focused on a fully parallel hardware implementation. In the case of software implementation on modern computers, when multiplication and addition can take about the same time, the proposed algorithm does not give a gain in speed.

Table 1. Number of multiplications and additions of real numbers needed to calculate the rotated vector.

type of operation	quaternion multiplication	rotation matrix	proposed algorithm
$\times$	24	19	14
$+$	17	21	43

6. Conclusion

We presented a new algorithm for quaternion-based spatial rotation. During the synthesis of the discussed algorithm we use the fact that the quaternion-based spatial rotation may be represented as the matrix-vector product $\mathbf{y}_4 = \mathbf{Q}_4\mathbf{Q}_4^*\mathbf{x}_4$. The matrices participating in the product have unique structural properties that allow performing its advantageous decompositions. Namely, these decompositions lead to reducing the multiplicative complexity of computations. If we compare the effectiveness of the discussed two approaches, we can conclude the following. The algorithm using the rotation matrix is more efficient when the set of points (vectors) of a solid body performs the same rotation, because in this case the rotation matrix is calculated only once and is the same for all points (vectors) of this body. But if we are dealing with a set of bodies or objects which could be treated as ideal particles and rotate at different angles or around different axes, the rotation matrix must be calculated for each element separately and then the algorithm using this matrix loses in comparison with our algorithm. In particular, such a case occurs when we are dealing with a swarm of unmanned aerial vehicles, each of which performs an independent movement and is controlled from a common center (Alfeo *et al.*, 2018; Choutri *et al.*, 2018; Avellar *et al.*, 2015; Joordens and Jamshidi, 2009). Another effective application of our approach is quaternion-based information security (Czaplewski *et al.*, 2014; Dzwonkowski *et al.*, 2015; Czaplewski and Rykaczewski, 2015). Thus, in some cases the presented algorithm may appear to be more applicable and convenient from the implementation point of view.

References

Abbena, E., Salamon, S. and Gray, A. (2006). *Modern Differential Geometry of Curves and Surfaces with Mathematica, 3rd Edition*, Chapman & Hall/CRC, Boca Raton, FL.

Alfeo, A., Cimino, M., Francesco, N.D., Lazzeri, A., Lega, M. and Vaglini, G. (2018). Swarm coordination of mini-UAVs for target search using imperfect sensors, *Intelligent Decision Technologies* **12**(2): 149–162.

Alfsmann, D., Göckler, H.G., Sangwine, S.J. and Ell, T.A. (2007). Hypercomplex algebras in digital signal

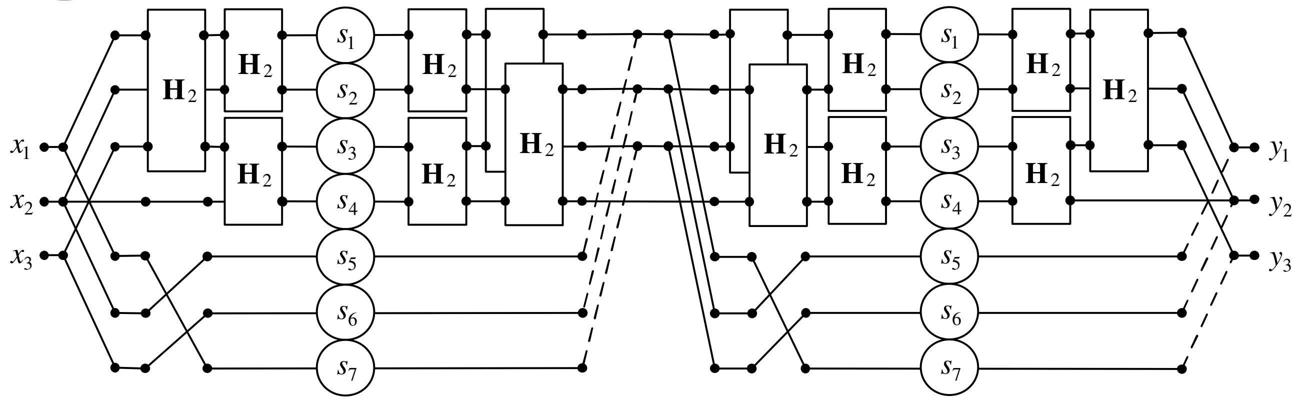

Fig. 2. Data flow diagram for a rationalized quaternion-based 3D rotation algorithm.

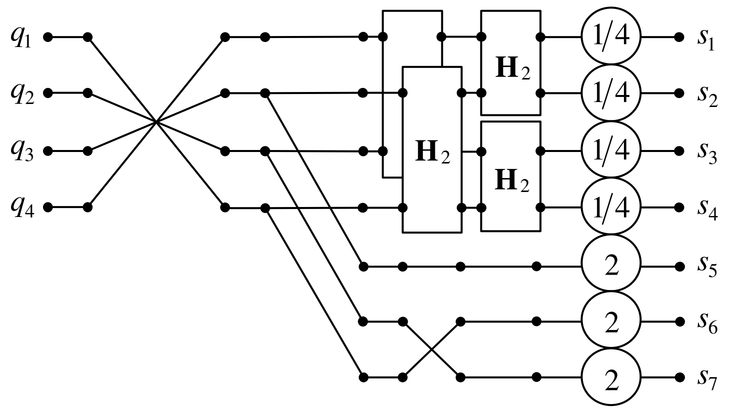

Fig. 3. Data flow diagram describing the process of calculating entries of the matrix $\mathbf{D}_7$.

processing: Benefits and drawbacks (tutorial), *EURASIP 15th European Signal Processing Conference (EUSIPCO 2007), Poznań, Poland*, pp. 1322–1326.

Andreis, D. and Canuto, E.S. (2004). Orbit dynamics and kinematics with full quaternions, *Proceedings of the 2004 American Control Conference, Boston, MA, USA*, pp. 3660–3665.

Angel, E. and Shreiner, D. (2012). *Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL, 6th Edition*, Pearson, Harlow.

Avellar, G., Pimenta, G.P.L. and Iscold, P. (2015). Multi-UAV routing for area coverage and remote sensing with minimum time, *Sensors* **15**(11): 27783–27803.

Bayro-Corrochano, E. (2006). The theory and use of the quaternion wavelet transform, *Journal of Mathematical Imaging and Vision* **24**(1): 19–35.

Bülow, T. and Sommer, G. (2001). Hypercomplex signals—a novel extension of the analytic signal to the multidimensional case, *IEEE Transactions on Signal Processing* **49**(11): 2844–2852.

Çakir, M. and Bütün, E. (2010). Constrained trajectory planning for cooperative work with behavior based genetic algorithm, *in* A. Ponce de Leon F. de Carvalho *et al.* (Eds), *Computing and Artificial Intelligence*, Advances in Intelligent and Soft Computing, Vol. 79, Springer, Berlin/Heidelberg, pp. 497–508.

Cariow, C. (2014). Strategies for the synthesis of fast algorithms for the computation of the matrix-vector products, *Journal of Signal Processing Theory and Applications* **3**(1): 1–19.

Choutri, K., Lagha, M., Dala, L. and Lipatov, M. (2018). Quadrotors UAVs swarming control under leader-followers formation, *22nd International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania*, pp. 794–799.

Czaplewski, B., Dzwonkowski, M. and Rykaczewski, R. (2014). Digital fingerprinting based on quaternion encryption scheme for gray-tone images, *Journal of Telecommunications and Information Technologies* **2**: 3–11.

Czaplewski, B. and Rykaczewski, R. (2015). Receiver-side fingerprinting method for color images based on a series of quaternion rotations, *Przegląd Telekomunikacyjny + Wiadomości Telekomunikacyjne* (8–9): 1127–1134.

Dzwonkowski, M., Papaj, M. and Rykaczewski, R. (2015). A new quaternion-based encryption method for DICOM images, *IEEE Transactions on Image Processing* **24**(11): 4614–4622.

Ell, T.A. (1993). Quaternion-Fourier transforms for analysis of two-dimensional linear time-invariant partial differential systems, *Proceedings of the 32nd IEEE Conference on Decision and Control, San Antonio, TX, USA*, Vol. 2, pp. 1830–1841.

Fenwick, E.H. (1992). Quaternions and the art of navigation, *International Journal of Mathematical Education in Science and Technology* **23**(2): 273–279.

Fresk, E. and Nikolakopoulos, G. (2013). Full quaternion based attitude control for a quadrotor, *European Control Conference (ECC), Zürich, Switzerland*, pp. 3864–3869.

Funda, J., Taylor, R. and Paul, R. (1990). On homogeneous transforms, quaternions, and computational efficiency, *IEEE Transactions on Robotics and Automation* **6**(3): 382–388.

Goldstein, H. (1980). Finite rotations, *Classical Mechanics*, 2nd Edn, Addison-Wesley, Reading, MA, pp. 164–166.

Hanson, A.J. (2006). *Visualizing Quaternions*, Series in Interactive 3D Technology, Morgan Kaufmann Publishers, San Francisco, CA.

Hu, C., Meng, M. Q.-H., Mandal, M. and Liu, P.X. (2006). Robot rotation decomposition using quaternions, *Proceedings of the 2006 International Conference on Mechatronics and Automation, Luoyang, Henan, China*, pp. 1158–1163.

Huynh, D. (2009). Metrics for 3D rotations: Comparison and analysis, *Journal of Mathematical Imaging and Vision* **35**(2): 155–164.

Jia, Y.-B. (2015). Quaternions and rotations, *Lecture Notes*, Iowa State University, Ames, IO, `http://web.cs.iastate.edu/~cs577/handouts/quaternion.pdf`.

Joordens, M. and Jamshidi, M. (2009). Underwater swarm robotics consensus control, *2009 IEEE International Conference on Systems, Man and Cybernetics, San Antonio, TX, USA*, pp. 3163–3168.

Kanade, T. (1987). *Three-Dimensional Machine Vision*, Springer International Series in Engineering and Computer Science, Vol. 21, Kluwer Academic Publishers, Norwell, MA.

Kantor, I.L. and Solodovnikov, A.S. (1989). *Hypercomplex Numbers*, Springer-Verlag, New York, NY.

Kuipers, J.B. (1999). *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*, Princeton University Press, Princeton, NJ.

Kunze, K. and Schaeben, H. (2004). The Bingham distribution of quaternions and its spherical radon transform in texture analysis, *Mathematical Geology* **36**(8): 917–943.

Lengyel, E. (2011). *Mathematics for 3D Game Programming and Computer Graphics, 3rd Edition*, Course Technology Cengage Learning PTR, Boston, MA.

Malekian, E., Zakerolhosseini, A. and Mashatan, A. (2011). QTRU: Quaternionic version of the NTRU public-key cryptosystems, *ISC International Journal of Information Security* **3**(1): 29–42.

Markley, F.L. (2008). Unit quaternion from rotation matrix, *Journal of Guidance, Control, and Dynamics* **31**(2): 440–442.

Mukundan, R. (2002). Quaternions: From classical mechanics to computer graphics, and beyond, *Proceedings of the 7th Asian Technology Conference in Mathematics, Melaka, Malaysia*, pp. 97–106.

Nüchter, D. (2009). *Robotic Mapping*, Springer Tracts in Advanced Robotics, Vol. 52, Springer-Verlag, Berlin/Heidelberg, pp. 35–76.

Pei, S.-C., Ding, J.-J. and Chang, J. (2001). Color pattern recognition by quaternion correlation, *Proceedings of the 2001 International Conference on Image Processing, Thessaloniki, Greece*, Vol. 1, pp. 894–897.

Pei, S.-C., Ding, J.-J. and Chang, J.-H. (2011). Efficient implementation of quaternion Fourier transform, convolution, and correlation by 2-d complex FFT, *IEEE Transactions on Signal Processing* **49**(11): 2783–2797.

Pletinckx, D. (1989). Quaternion calculus as a basic tool in computer graphics, *The Visual Computer* **5**(2): 2–13.

Roberts, G.N., Sutton, R. and Ye, M. (2006). *Advances in Unmanned Marine Vehicles*, Institution of Engineering and Technology, Stevenage.

Rousseau, P., Desrochers, A. and Krouglicof, N. (2002). Machine vision system for the automatic identification of robot kinematic parameters, *IEEE Transactions on Robotics and Automation* **17**(6): 972–978.

Sangwine, S.J. and Bihan, N.L. (2007). Hypercomplex analytic signals: Extension of the analytic signal concept to complex signals, *EURASIP 15th European Signal Processing Conference (EUSIPCO 2007), Poznań, Poland*, pp. 621–624.

Shuster, M.D. and Natanson, G.A. (1993). Quaternion computation from a geometric point of view, *The Journal of Astronautical Sciences* **41**(4): 545–556.

Steeb, W.-H. and Hardy, Y. (2011). *Matrix Calculus and Kronecker Product: A Practical Approach to Linear and Multilinear Algebra, 2nd Edition*, World Scientific Publishing Company, Singapore.

Terzakis, G., Culverhouse, P., Bugmann, G., Sharma, S. and Sutton, R. (2014). On quaternion based parameterization of orientation in computer vision and robotics, *Journal of Engineering Science and Technology Review* **7**(1): 82–93.

Vicci, L. (2001). Quaternions and rotations in 3-space: The algebra and its geometric interpretation, *Technical Report TR01-014*, University of North Carolina at Chapel Hill, Chapel Hill, NC, `http://www.cs.unc.edu/techreports/01-014.pdf`.

Vince, J. (2011). *Quaternions for Computer Graphics, 1st Edition*, Springer, London.

Wareham, R., Cameron, J. and Lasenby, J. (2005). Applications of conformal geometric algebra in computer vision and graphics, *in* H. Li *et al.* (Eds), *Computer Algebra and Geometric Algebra with Applications*, Springer-Verlag, Berlin/Heidelberg, pp. 329–349.

Witten, B. and Shragge, J. (2006). Quaternion-based signal processing, *Proceedings of the New Orleans Annual Meeting, New Orleans, LA, USA*, pp. 2862–2865.

Wysocki, B.J., Wysocki, T.A. and Seberry, J. (2006). Modeling dual polarization wireless fading channels using quaternions, *Joint IST Workshop on Mobile Future and the Symposium on Trends in Communications SympoTIC'06, Bratislava, Slovakia*, pp. 68–71.

Aleksandr Cariow received his PhD and DSc degrees in computer science from LITMO University in St. Petersburg, Russia, in 1984 and 2001, respectively. In 1999, he joined the Faculty of Computer Science and Information Technology, West Pomeranian University of Technology in Szczecin, Poland, where he currently works in the Department of Computer Architectures and Telecommunications. His research interests include digital signal processing algorithms, VLSI architectures, and data processing parallelization.

Galina Cariowa received the MSc degree in mathematics from Moldavian State University, Chişinău, in 1978 and the PhD degree in computer science from the West Pomeranian University of Technology in Szczecin, Poland, in 2007. She is currently working as an assistant professor at the Department of Computer Architectures and Telecommunications there. Her scientific interests include numerical linear algebra and digital signal processing algorithms, VLSI architectures, and data processing parallelization.

Dorota Majorkowska-Mech received the MSc degree in physics in 1994 and the MSc degree in mathematics in 1996 from the University of Szczecin. In 2008 she obtained the PhD degree in computer science from the West Pomeranian University of Technology in Szczecin, where she currently works in the Department of Computer Architectures and Telecommunications. She is interested in algorithms of discrete orthogonal transforms.

Received: 21 March 2019
Revised: 22 July 2019
Re-revised: 10 October 2019
Accepted: 18 October 2019