Quaternions - Redundancy + Efficiency = Ternions

Ulrich Mutze
ulrichmutze@aol.com

The computational implementation of rotations in three-dimensional space based on quaternions is modified such that three-component objects without constraints play the role of the four-component constraint quaternions. It is shown that this increases speed and accuracy significantly.

1 Introduction

To describe how rigid bodies rotate in space is an important task both in computational physics as in computer graphics. Especially in the last field, convenience in programming and efficieny in execution made the usage of quaternions for this purpose popular. In order to make my modification of this method clear, I'll go through the defining elements of the quaternion method:

A quaternion is an object which is uniquely given by a list of four real numbers (w,x,y,z) (an element of $\mathbb{R}^4$) and for which, based on these components, arithmetical operations are understood to be defined as follows

$$
\begin{aligned}
(w_1,x_1,y_1,z_1)+(w_2,x_2,y_2,z_2) &:= (w_1+w_2\,,\,x_1+x_2\,,\,y_1+y_2\,,\,z_1+z_2)\\
(w,x,y,z)\cdot a &:= (wa\,,\,xa\,,\,ya\,,\,za)\,,\quad a\in\mathbb{R}\\
(w_1,x_1,y_1,z_1)\circ(w_2,x_2,y_2,z_2) &:= (w_1w_2-x_1x_2-y_1y_2-z_1z_2\,,\,w_1x_2+x_1w_2+y_1z_2-z_1y_2\,,\\
&\qquad w_1y_2-x_1z_2+y_1w_2+z_1x_2\,,\,w_1z_2+x_1y_2-y_1x_2+z_1w_2\,)\\
\overline{(w,x,y,z)} &:= (w\,,\,-x\,,\,-y\,,\,-z)\,,\quad \mathcal{N}(w,x,y,z) := w^2+x^2+y^2+z^2\\
(w,x,y,z)^{-1} &:= \overline{(w,x,y,z)}\cdot\frac{1}{\mathcal{N}(w,x,y,z)}\,,\quad \|(w,x,y,z)\| := \sqrt{\mathcal{N}(w,x,y,z)}
\end{aligned}
$$

Here we are not concerned with the whole non-commutative division algebra of quaternions but only with the subgroup of *normalized* quaternions, i.e of quaternions q for which $\|q\|=1$. Only these quaternions are related to rotation in 3-space, and only to them the title of this paper refers.

For any normalized quaternion q and any vector $v = (v_1, v_2, v_3)$ the image $v' = q \cdot v = (v'_1, v'_2, v'_3)$ of v under the action of q is defined by

$$q \circ (a, v_1, v_2, v_3) \circ \overline{q} = (a, v'_1, v'_2, v'_3) \,, \tag{1}$$

where a is an arbitrary real number for which here 0 is the most natural choice, but for which users of graphics libraries might expect to see 1. There are two essential facts about this:

- The mapping $v \mapsto v'$ is a 3D rotation, and for the rotation angle φ and the unit vector $n = (n_1, n_2, n_3)$ of the rotation axis, there is the connection

$$q = (\cos(\varphi/2)\,,\ n_1 \sin(\varphi/2)\,,\ n_2 \sin(\varphi/2)\,,\ n_3 \sin(\varphi/2)) \tag{2}$$

- For a sequence of normalized quaternions acting on a vector one has the chaining law

$$(q_1 \circ q_2 \circ \ldots \circ q_n) \cdot v = q_1 \cdot (q_2 \cdot (\ldots (q_n \cdot v) \ldots)) \tag{3}$$

The right-hand side of this equation is what arrises most directly in applications dealing with motion of rigid bodies or with changes of reference systems. The equality to the left-hand side then provides us with a computationally cheaper equivalent. If the same chain of transformations is to be applied to more than one vector, the benefit of computing $q := q_1 \circ q_2 \circ \ldots \circ q_n$ once, and applying it to the various vectors, is even larger. Notice that round-off errors will prevent q from being exactly normalized when the product is coded verbatim. An autonomous algorithm for the multiplication of arbitrarily many normalized quaternions has to specify a small number ε, and to test for each result q_r of a multiplication whether $|\mathcal{N}(q_r) - 1| < \varepsilon$. If this is not the case, it has to replace q_r by $q_r \cdot \mathcal{N}(q_r)^{-1/2}$, a step that I will refer to as *renormalization*. It is this improved multiplication law of normalized quaternions which will be used in the computational experiment in section 3. Replacing normalized quaternions by ternions eliminates the need for renormalization, but also needs to specify an ε to prevent the multiplication law from failing in rare cases.

Actually, an efficient implementation of rotations will not use equation (1) as it stands, but only after having brought it into the form of a transformation matrix: For the normalized quaternion $q = (q_0, q_1, q_2, q_3)$ one forms the 3×3-matrix

$$Q := \begin{pmatrix} {q_0}^2 + {q_1}^2 - 1/2 & q_1 q_2 + q_0 q_3 & q_1 q_3 - q_0 q_2 \\ q_1 q_2 - q_0 q_3 & {q_0}^2 + {q_2}^2 - 1/2 & q_2 q_3 + q_0 q_1 \\ q_1 q_3 + q_0 q_2 & q_2 q_3 - q_0 q_1 & {q_0}^2 + {q_3}^2 - 1/2 \end{pmatrix} \tag{4}$$

and finally the rotated vector $v' = q \cdot v$ by

$$v'_i := 2(Q_{i1} v_1 + Q_{i2} v_2 + Q_{i3} v_3) \,, \quad i = 1, 2, 3 \,. \tag{5}$$

Since a quaternion product needs 16 multiplications and 12 addition/subtraction operations (16M+12A for shortness), the operation count of (1) is 32M+24A. The implementation (4),(5) needs only 22M+18A, and even only 12M+6A if the transformation matrix does not change from vector to vector.

2 Ternions

Let (q_0, q_1, q_2, q_3) be the generic normalized quaternion, then $(q_1/q_0, q_2/q_0, q_3/q_0)$ is the corresponding *ternion* and it is understood that the arithmetic operations are carried over to the ternions by just this correspondence. Since q_0 is zero for quaternions that correspond to rotations for which the rotation angle φ equals π, one could expect to run into mathematical and numerical singularity problems in carrying this out. As we will see, such problems don't exist: ternions do all what normalized quaternions do at a significantly reduced operations count and without posing a renormalization problem. The following exposition will not deduce the ternion laws from their quaternion analogs. Instead, they will be simply stated and the origin will be obvious.

A ternion is an object which is uniquely given by a list of three real numbers (x,y,z) (an element of $\mathbb{R}^3$) and for which, based on these components, arithmetical operations are understood to be defined as follows

$$\begin{aligned}(x,y,z)\cdot a &:= (xa\,,\ ya\,,\ za)\,, \quad a\in\mathbb{R}\\ (x_1,y_1,z_1)\circ(x_2,y_2,z_2) &:= (x_1+x_2+y_1z_2-z_1y_2\,,\\ &\qquad y_1+y_2+z_1x_2-x_1z_2\,,\\ &\qquad z_1+z_2+x_1y_2-y_1x_2)\cdot S\end{aligned} \tag{6}$$

where S is defined by the following rule:

$$\begin{aligned}s &:= 1-x_1x_2-y_1y_2-z_1z_2\,, \quad S := |s|<\varepsilon\,?\quad 1/\varepsilon \quad : \quad 1/s\\ (x,y,z)^{-1} &:= (-x\,,\ -y\,,\ -z)\end{aligned}$$

Unlike the corresponding parameter for normalized quaternions, the parameter ε does not control a speed/accuracy trade-off. It simply avoids division by zero, and for practical puposes all values $< 10^{-6}$ have the same effect. The graphs in section 3 cover the range $10^{-3} \le \varepsilon \le 3\cdot 10^{-22}$ and show no influence of ε.

Let $r = (r_1, r_2, r_3)$ the generic ternion and $v = (v_1, v_2, v_3)$ a generic vector. Then we define $v' = r\cdot v$ similar to equations (4),(5):

$$R := \begin{pmatrix} {r_1}^2+b & r_1r_2+r_3 & r_1r_3-r_2\\ r_1r_2-r_3 & {r_2}^2+b & r_2r_3+r_1\\ r_1r_3+r_2 & r_2r_3-r_1 & {r_3}^2+b \end{pmatrix} \tag{7}$$

and finally the rotated vector $v' = q\cdot v$ by

$$v'_i := c(R_{i1}v_1+R_{i2}v_2+R_{i3}v_3)\,, \quad i=1,2,3\,, \tag{8}$$

where the numbers b and c are defined by the following rule:

$$\begin{aligned}\rho &:= {r_1}^2+{r_2}^2+{r_3}^2\\ b &:= (1-\rho)/2\\ c &:= 2/(1+\rho)\,.\end{aligned} \tag{9}$$

Equation (2) becomes

$$r = (n_1 \tan(\varphi/2)\,,\; n_2 \tan(\varphi/2)\,,\; n_3 \tan(\varphi/2)) \tag{10}$$

and equation (3) remains valid with q replaced by r.

3 A computational comparison

This describes a computational experiment designed to measure the gain in speed and accuracy associated with going from quaternions to ternions. The experiment follows a poven pattern by setting up a huge computation with randomly selected objects which—by mathematical reasons that can't be anticipated by an optimizing compiler—has the exact result 0. Since each of the methods under consideration has its *method parameter* ε, we vary this parameter over a wide range ($10^{-3} \le \varepsilon \le 3 \cdot 10^{-22}$) for each method. Beware that the effect of ε is quite different in the two methods, so that using their numerical value as a common abscissa in the figures would be missleading if the ternion data would not turn out to be independent of this value.

We generate a random sequence $q_1, q_2, \ldots, q_n$ of normalized quaternions and the corresponding sequence $r_1, r_2, \ldots, r_n$ of ternions ($n = 2.5 \cdot 10^6$). The rotation angle is uniformely distributed in $[\pi - 10^{-6}, \pi + 10^{-6}]$ in order to see problems with π-rotations if there are some. Further, we generate a random sequence $v_1, v_2, \ldots, v_m$ of vectors ($m = 10^3$) with components uniformely distributed in $[-1, 1]$. The following steps will be described for quaternions only with the understanding that the corresponding steps will be carried out for ternions: We compute $q_0 := q_1 \circ q_2 \circ \ldots \circ q_n \circ {q_n}^{-1} \circ \ldots \circ {q_2}^{-1} \circ {q_1}^{-1}$ and memorize the midway result $q_c := q_1 \circ q_2 \circ \ldots \circ q_n$. The exact result for q_0 is the zero rotation and the computational result is a small rotation that can be quantified by its rotation angle φ. I refer to this quantity as the *algebraic error*. Then we apply q_c to all m vectors v_i and get the vectors ${v'}_i$. A convenient measure for the deviation of this mapping from a rotation is

$$\delta := \sqrt{\sum_{i<j} (|{v'}_i - {v'}_j| - |v_i - v_j|)^2 + \frac{m}{2} \sum_i (|{v'}_i| - |v_i|)^2}$$

This quantity will be referred to as the *geometric error*.

Figure 1 shows δ as a function of the method parameter ε. The symbols ■ and ▲ always mark computed values referring to the quaternion part and the ternion part of the program respectively. As mentioned already, for ternions the data show no significant dependence on ε. Moreover, it is very low and only due to numerical noise. For quaternions the deformation is five orders of magnitude larger. Only when ε is small enough that most of the computed quaternion products get renormalized, the deformation gets reduced considerably to roughly the ternion value.

Figure 2 shows the φ error in the same style. Here we see that both methods satisfy the algebraic identity under consideration up to numerical noise. That the onset

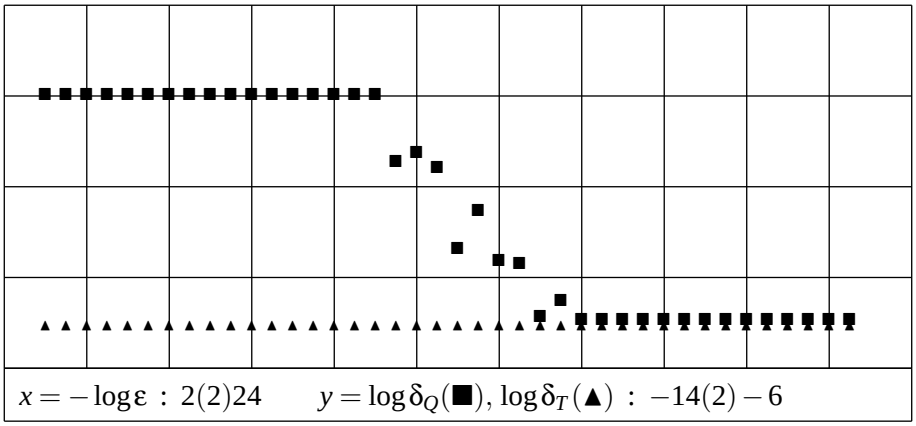

Figure 1: Comparison of geometric errors

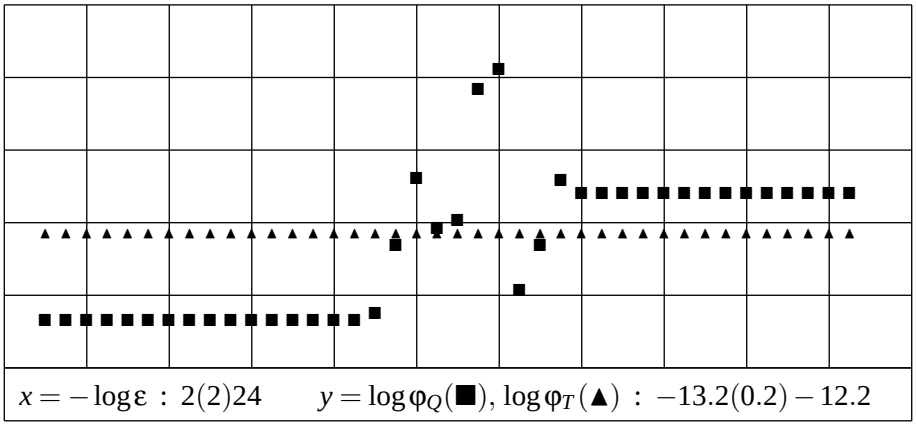

Figure 2: Comparison of algebraic errors

of normalization changes the noise level is natural. The tendency of this change is accidental. That ternions hold a very low error level consistenly renders hidden flaws of the method very unlikely.

Finally, figure 3 shows the computation time (in seconds) spent in my program run on quaternion/ternion multiplications and vector transformations (thus excluding the random generation of objects, the computations of the error measures, and writing results to files; in effect it is the time for 5 million multiplications of (qua)ternions). The computation was done on a off-the-shelf computer (2.08 GHz, OS Linux, C++, compiler g++ 3.3.1). Since I was not able to separate the time spent on the program from time spent on other system activities, the values shown here are minimum values over 20 automatic repetitions of the same program part (preferring the minimum value over the mean value is natural if—so my hypothesis—the bad reproducibility of internal

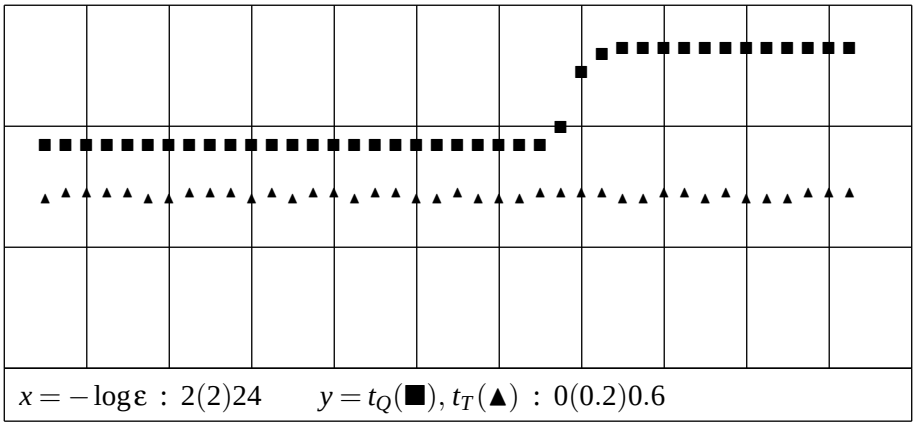

Figure 3: Comparison of execution times

clock readings is due to stochastic time consuming events). The data on which the figure is based figure shows that there are only four times in the game when the transition zone in the quaternion data is ignored: $t_{Q1} = 0.37, t_{Q2} = 0.53, t_{T1} = 0.28, t_{T2} = 0.29$ (no more digits, this is not a formatting artifact!) According to the hypothesis stated above 0.28*s* is the measured value for the computation time of the ternion method. So the speed advantage of the ternion method was found as $0.37/0.28 = 1.32$ in the non-renormalization regime and $0.53/0.28 = 1.90$ in the renormalization regime. Counting additions, subtraction, multiplications, divisions and square-roots with equal weight, one would expect $36/25 = 1.44$ and $42/25 = 1.68$ instead of this. So that we measured $0.44/0.32 = 73\%$ of the expected speed advantage in one regime and $0.90/0.68 = 132\%$ of the expected speed advantage in the other regime. Thus the picture is not clear in all detail, but a significant speed advantage of the ternion method is shown to exist.

Acknowledgments

I'm grateful to Professor emeritus Hans Joachim Meister, who used $\tan(\varphi/2)$ to express composition of rotations in his lectures on group representations in physics in 1965 at the University of Munich (LMU). Consulting his lecture notes triggered the idea for the present method in 1986, when I used it first in an astronomical program.

Postscript

Thomas Dera made me aware of the publication [1] which shows that what I considered a new trick is actually a time-honoured discovery that, however, is not as widely known as it should be. What is called *ternions* in the present article is called *rotation vectors* in [1] from which the following citation is taken

> The development of this parameterization of rotations can probably be attributed to Rodrigues (1840) ([2]) and the coefficients of the rotation vectors are sometimes referred to as Euler-Rodrigues parameters [3]. One of the first to rediscover these parameters for the oculomotor field was Haustein (1989) ([4]), whose paper also provides a good introduction to rotation vectors.

The references within this citation have been converted verbatim to entries in the following list.

References

[1] Thomas Haslwanter: Mathematics of Three-dimensional Eye Rotations, Vision Res. Vol 35, No 12, pp 1727-1739, 1995

[2] Rodrigues, O. (1840). Des lois géométriques qui régissent les déplacements d'un système solide dans l'espace, et de la variation des coordonnées provenant de ses déplacements considérés indéndamment des causes qui peuvent les produire. *Journal de Mathématique Pures et Appliquées, 5*, 380-440.

[3] Altmann,S.L.(1986). *Rotations, quaternions and double groups*. Oxford: Clarendon Press.

[4] Haustein, W. (1989). Considerations on Listing's Law and the primary position by means of a matrix description of eye position control. *Biological Cybernetics, 60*, 411-420.