A Beginners Guide to Dual-Quaternions

What They Are, How They Work, and How to Use Them for 3D Character Hierarchies

Ben Kenwright
School of Computing Science, Newcastle University
Newcastle Upon Tyne, United Kingdom
b.kenwright@ncl.ac.uk

ABSTRACT

In this paper, we give a beginners guide to the practicality of using dual-quaternions to represent the rotations and translations in character-based hierarchies. Quaternions have proven themselves in many fields of science and computing as providing an unambiguous, un-cumbersome, computationally efficient method of representing rotational information. We hope after reading this paper the reader will take a similar view on dual-quaternions. We explain how dual number theory can extend quaternions to dual-quaternions and how we can use them to represent rigid transforms (i.e., translations and rotations). Through a set of examples, we demonstrate exactly how dual-quaternions relate rotations and translations and compare them with traditional Euler's angles in combination with Matrix concatenation. We give a clear-cut, step-by-step introduction to dual-quaternions, which is followed by a no-nonsense how-to approach on employing them in code. The reader, I believe, after reading this paper should be able to see how dual-quaternions can offer a straightforward solution of representing rigid transforms (e.g., in complex character hierarchies). We show how dual-quaternions propose a novel alternative to pure Euler-Matrix methods and how a hybrid system in combination with matrices results in a faster more reliable solution. We focus on demonstrating the enormous rewards of using dual-quaternions for rigid transforms and in particular their application in complex 3D character hierarchies.

Keywords

Dual-Quaternion, 3D, Real-Time, Character Hierarchies, Rigid Transformation

1. INTRODUCTION

Real-time dynamic 3D character systems combine key framed animations, inverse kinematics (IK) and physics-based models to produce controllable, responsive, realistic motions. The majority of character-based systems use a skeleton hierarchical composition of rigid transforms. Each rigid transform has six degrees of freedom (DOF) that consists of three translational and three rotational components. Matrices are the most popular method of storing and combining these transforms. While matrices are adequate, we ask the question, is there a better method? In this paper, we address the advantages and disadvantages of matrices while proposing a novel alternative based on quaternions called dual-quaternions. The purpose of this paper is to present a beginner's guide to dual-quaternions in sufficient detail that the reader can begin to use them as a practical problem-solving tool for rigid character transforms. This paper covers the basics of dual-quaternions and their application to complex hierarchical systems with many DOF.

Dual-quaternions are interesting and important because they cut down the volume of algebra. They make the solution more straightforward and robust. *They allow us to unify the translation and rotation into a single state*; instead of having to define separate vectors. While matrices offer a comparable alternative to dual-quaternions, we argue that they can be inefficient and cumbersome in comparison. In fact, dual-quaternions give us a compact, un-ambiguous, singularity-free, and computational minimalistic rigid transform. In addition, dual-quaternions have been shown to be the most efficient and most compact form of representing rotation and translation. Dual-quaternions can easily take the place of matrices in hierarchies at no additional cost. For rigid transform hierarchies that combine and compare rigid transforms on a frame-by-frame bases (e.g., character inverse kinematics (IK) and joint constraints), alternative methods such as matrices need to be converted to quaternions to generate reliable contrast data; this can be done without any conversion using dual-quaternions.

Many students have a great deal of trouble understanding essentially what quaternions are and how they can represent rotation. So when the subject of dual-quaternions is presented, it is usually not welcomed with open arms. Dual-quaternions are a break from the norm (i.e., matrices) which we hope to entice the reader into embracing to represent their

rigid transforms. The reader should walk away from this paper with a clear understanding of what dual-quaternions are and how they can be used.

The majority of computer scientists are familiar with vectors, matrices, and quaternions. They provide a set of tools to help solve problems. This paper presents the case for adding dual-quaternions to this set of tools.

The contribution of this paper is the explanation and demonstration of dual-quaternions in a sufficiently detailed way that the reader can begin to appreciate their practical problem-solving advantages. We use character-based hierarchies as a base method to illustrate the realistic reward of dual-quaternions in time critical systems (e.g., games).

This paper presents dual-quaternions as a method for representing rigid transforms in complex character hierarchies with a large number of DOF. We explain how to implement a basic dual-quaternion class and combine dual-quaternions through straightforward multiplication to work in place of matrices.

The roadmap for the rest of the paper is as follows: we begin with a review of recent and related work that emphasises the power of dual-quaternions. We review familiar rigid transform methods and their advantages and disadvantages. We then outline the primary reasons for using dual-quaternions and why you would want to use them for rigid transforms over other methods. We then give the background mathematical information for dual numbers, quaternions and dual-quaternions. The following sections then focus on the practical aspects of dual-quaternions. We discuss a variety of experiments with computer simulations and character hierarchies in relation to dual-quaternion. Finally, the end section presents the conclusion and proposed future work.

2. RELATED WORK

The dual-quaternion has been around since 1882 [CLIF82] but has gained less attention compared to quaternions alone. Comparable to quaternions the dual-quaternions have had a taboo associated with them, whereby students avoid quaternion and hence dual-quaternions. While the robotics community has started to adopt dual-quaternions in recent years, the computer graphics community has not embraced them as whole-heartedly. We review some recent work which has taken hold and has demonstrated the practicality of dual-quaternions, both in robotics and computer graphics.

2.1. Computer Graphics

Kavan [KCŽO08] demonstrated the advantages of dual-quaternions in character skinning and blending.

Ivo [IVIV11] extended Kavans [KCŽO08] work with dual-quaternions and qtangents as an alternative method for representing rigid transforms instead of matrices, and gives evidence that the results can be faster with accumulated transformations of joints if the inferences per vertex are large enough.

Selig [SELI11] address the key problem in computer games. Examining the problem of solving the equations of motion in real-time and puts forward how dual-quaternion give a very neat and succinct way of represent rigid-body transformations.

Vasilakis [VAFU09] discussed skeleton-based rigid-skinning for character animation.

Kuang [KMLX11] presented a strategy for creating real-time animation of clothed body movement.

2.2. Robotics

Pham [PPAF10] solved linked chain inverse kinematic (IK) problems using Jacobian matrix in the dual-quaternion space.

Malte [SCHI11] used a mean of multiple computational (MMC) model with dual-quaternions to model bodies.

Ge [GVMC98] demonstrated dual-quaternions to be an efficient and practical method for interpolating three-dimensional motions.

Yang-Hsing [LIWC10] calculated the relative orientation using dual-quaternions.

Perez [PEMC04] formulated dynamic constraints for articulated robotic systems using dual-quaternions.

3. FAMILIAR PHYSICAL CONCEPTS

We review the most common methods of representing rigid body orientations and translations in our physical world (three spatial dimensions). While orientation and rotation are familiar concepts, there are many ways to represent them both mathematically and computationally, each with their own strengths and weaknesses. We briefly describe four of the most popular methods of representing rigid transforms in character systems. This helps illustrate the mathematical and computational issues that occur. The four alternate methods we compare mathematically and computationally to dual-quaternions are:

- Matrices
- Axis-Angles
- Euler-Angles
- Quaternions

} + Translation

Each alternative method needs to represent both the orientation and translation. In some cases this is achieved by using two separate state variables and combining them separately, while matrices and dual-quaternions give us a unified state variable.

For each case we focus on issues of interpolation, computational speed, mathematical robustness and distance metrics.

The properties we look for to represent the rigid body transform are:

Robustness – be continuous and not contain any discontinuities (such as gimbal lock with Euler's angles which we discuss later). Contain a unique representation, where some methods contain redundant information, such that several or an infinite number of elements can represent the same transform.

Efficiency – should consume the smallest necessary amount of space and be computationally fast. We would like a minimum number of calculations to combine and convert between alternative representations (e.g., cost to convert between matrices and Euler angles).

Ease of Use – can be used without too many complications.

3.1. Orientation and Translation

It might seem intuitive how objects are rotated and translated. For example, we can pick up any object around us and spin (rotate) and translate (move) it without thinking. However, how do we model this computationally and mathematically? The following sub-sections are devoted to the explanation and understanding of these basic principles.

For methods which are formed from separate orientation and translational information, we can analyse their workings by considering orientation and translation separately and combining them at the end of each transform.

3.2. Translation

The translation coordinates are relatively simple to work with. They compose of the scalar values along each of the principle axes (x, y and z). The computed orientations are combined with the translations by rotating the principle axis.

3.3. Euler-Angles

A familiar way of representing the orientation and translation in character systems is to factor it into three sequential angles around the principle orthogonal axes (x, y and z).

Euler's angles in 3D <u>do not</u> (in-general) *commute* under composition.

In practice, the angles are used by inserting them into matrices. The product of the three angle-matrices produces the Euler angle set. There are twelve possible products: XYZ, XYX, YZX, YZY, ZXY, ZXZ, XZY, XZX, YXZ, YXY, ZYX, and ZYZ. These are the order the rotations are applied in. For example, the factorization XYZ, would mean rotate round X then Y then Z.

To work with Euler angles we convert them to matrices:

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix}$$

$$\mathbf{Z} = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Combining the translation is just a matter of rotating the translational components (x, y and z) by the rotation.

To combine and calculate interpolating differences requires us to find the equivalent axis-angle of the two orientations and extrapolate the Euler angles.

- Create a matrix for each Euler angle.
- Multiply the three matrices together.
- Extract axis-angle from resulting matrix.

Converting, combining, and extracting Euler angles is computationally expensive. Moreover, Euler angles can have discontinuities around 0 and 2π, since the components live on separate circles and not a single vector space.

3.3.1. Advantages

People prefer Euler angles as they can comprehend them effortlessly and can create orientations with them without difficulty. They are also very intuitive and have a long history in physics and graphics and can make certain integrals over rotational space easier.

Euler angles are minimalistic and require only three parameters; however, we show later how four parameters are better than three. Furthermore, since the angles are used directly, there is no drifting or the need for normalization.

3.3.2. Disadvantages

Euler angles suffer from singularities - angles will instantaneously change by up to 2π radians as other angles go through the singularity; Euler angles are virtually impossible to use for sequential rotations. There are twelve different possible Euler angle rotation sequences - XYZ, XYX, XZY, and so on. There is no one "simplest" or "right" set of Euler angles. To derive a set of Euler angles you must know which rotational sequence you are using and stick to it.

In practice when Euler angles are needed; the underlying rotation operations are done using quaternions and are converted to Euler angles for the task at hand.

3.3.3. Gimbals Lock

The coordination singularity in Euler's angles is commonly referred to as gimbals lock. A gimbal is a physical device consisting of spherical concentric hoops with pivots connecting adjacent hoops, allowing them to rotate within each other (see Figure 1).

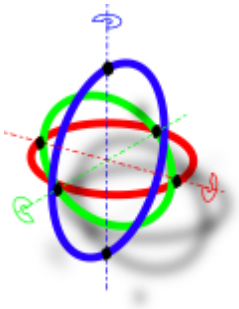

Figure 1. Gimbal with points of rotation indicated.

A gimbal is constructed by aligning three rings and attaching them orthogonally. Gimbals are often seen in gyroscopes used by the aeronautical industry.

As objects are rotated, they approach gimbal lock the singularity will cause numerical ill-conditioning, often evidented physically by the gimbal wiggling madly around as it operates near the singularity. This is one reason why the aerospace industry, early on, switched to quaternions to represent orientation – satellites, rockets and airplanes would have their navigation gyro lock up and could cause them to crash.

3.3.4. Interpolation

The major problem with Euler interpolation is that they have problems while interpolating near gimbals lock regions. When close to a gimbal lock singularity the interpolation become jittery and noise ridden; eventually becoming random and unstable as it converges on the singularity.

If Euler angles are interpolated linearly the resulting path will not take the shortest path between the endpoints as it does in vector space [ALMA92].

3.4. Axis-Angle

The axis-angle is represented by a unit axis and angle $(\hat{n},\theta)$ pair. This axis-angle representation can easily be converted to and from a matrix.

It is difficult to combine the axis-angle elements in their native form; usually being converted to an alternate representation for concatenation (e.g., matrices, quaternions).

3.4.1. Advantages

The greatest single advantage of the axis-angle representation is that it directly represents the action of rotation, while being straightforward and intuitive to work with.

3.4.2. Disadvantages

We can renormalize the axis since it is a unit vector, but numerical errors can still creep into the angle portion.

Infinite number of angle choices (multiples of 2π), so two axis-angle pairs can still refer to the same rotation but be different.

Axis-angle interpolation cannot be done using linear interpolation of the four elements. Interpolating between the four elements naively in this way does not give the shortest path.

Interpolating the angle alone can introduce discontinuities as the angle crosses from 0 to 2π. These 'jumps' are highly undesirable and can cause anarchy with the interpolation and numerical integration schemes.

3.5. Matrices

Representing a rigid transform using a matrix we extend a 3x3 rotation matrix to include translation information which makes it a 4x3 matrix. While a 4x3 matrix is the most efficient, on most occasions a 4x4 matrix is used because of availability.

The 3x3 part of the matrix consists of three orthogonal column vectors which are of unit magnitude.

A transform matrix can transform a vector coordinate by simply matrix multiplication:

$$\mathbf{y} = \mathbf{T}\,\mathbf{x}$$

where T is a transform matrix, x a vector coordinate and y the transformed result.

If the position and basis vectors are known, the transform matrix can trivially be produced, because each of the columns in the 3x3 part of the matrix represent the base vectors and the bottom row the translation.

The combination of matrix elements is achieved through simple multiplication. Matrices are not commutative and therefore their matrix representation of rigid body transforms is non-commutative as well.

3.5.1. Advantages

Matrices are taught in linear algebra early on in colleges so this makes them more familiar and favourable. In addition, a great many algorithms have been formulated and tested with matrices and so people choose them instinctively first.

3.5.2. Disadvantages

While matrices might seem to be the utopia, they in-fact can be found to have several problems.

Firstly, they take a minimum of 12 parameters to represent a structure with only six DOF; if memory is at a premium this can be undesirable.

Secondly, the rotational part of the matrix is composed of orthogonal columns which can drift and introduce unwanted scaling and sheering. We can re-normalize the matrix using Gram-Schmidt method [GILB86] but this can be computationally expensive.

Thirdly, interpolating between matrices is difficult. The three columns forming the orthogonal axis directions in the rotation part of the matrix do not represent the vector space and cannot be interpolated.

Finally, it is difficult to visualize a matrix and the axis-angle component about which it will rotate and translate.

3.6. Method Summary

We have outlined and examined current methods for representing a robust, practical and viable hierarchical rigid body solution. We now follow on from this by introducing and explaining how and why dual-quaternions stand-out above these methods.

4. WHY DUAL-QUATERNIONS?

We use dual-quaternions as a tool for expressing and analyzing the physical properties of rigid bodies. Dual-quaternions can formulate a problem more concisely, solve it more rapidly and in fewer steps, present the result more plainly to others, be put into practice with fewer lines of code and debugged effortlessly. Furthermore, there is no loss of efficiency; dual-quaternions can be just as efficient if not more efficient than using matrix methods. In all, there are several reasons for using dual-quaternions, which we summarize:

- Singularity-free
- Un-ambiguous
- Shortest path interpolation
- Most efficient and compact form for representing rigid transforms [SCHI11] - (3x4 matrix 12 floats compared to a dual-quaternion 8 floats)
- Unified representation of translation and rotation
- Can be integrated into a current system with little coding effort
- The individual translation and rotational information is combined to produce a single invariant coordinate frame [GVMC98]

5. DUAL NUMBERS

Clifford [CLIF82] introduced dual numbers; similar to complex numbers that consists of two parts known as the real and complex component. Dual numbers break the problem into two components and are defined as:

$$z = r + d\varepsilon \text{ with } \varepsilon^2 = 0 \text{ but } \varepsilon \neq 0$$

where ε is the dual operator, r is the real part and d the dual part. Similar to complex number theory, where i is added to distinguish the real and complex components, the dual operator ε is used in the same way.

The dual number theory can be extended to other concepts, such as vectors and real numbers, but we focus on their applicability in conjunction with quaternions to represent rotation and translation transforms.

5.1. Dual Number Arithmetic Operations

Dual numbers can perform the fundamental arithmetic operations below:

Addition

$$(r_A + d_A\varepsilon) + (r_B + d_B\varepsilon) = (r_A + r_B) + (d_A + d_B)\varepsilon$$

Multiplication

$$\begin{aligned}(r_A + d_A\varepsilon)(r_B + d_B\varepsilon) &= r_A r_B + r_A d_B\varepsilon + r_B d_A\varepsilon + d_A d_B\varepsilon^2 \\ &= r_A r_B + (r_A d_B + r_B d_A)\varepsilon\end{aligned}$$

Division

$$\begin{aligned}\frac{(r_A + d_A\varepsilon)}{(r_B + d_B\varepsilon)} &= \frac{(r_A + d_A\varepsilon)}{(r_B + d_B\varepsilon)}\frac{(r_B - d_B\varepsilon)}{(r_B - d_B\varepsilon)} \\ &= \frac{r_A r_B + (r_B d_A - r_A d_B)\varepsilon}{(r_B)^2} \\ &= \frac{r_A r_B}{r_B^2} + \frac{r_B d_A - r_A d_B}{r_B^2}\varepsilon\end{aligned}$$

Further reading on the subject of dual numbers is presented by Gino [BERG09].

5.2. Dual Number Differentiation

Dual numbers differentiate in the same way as any other vector using elementary calculus principles, e.g.:

$$\frac{\mathrm{d}}{\mathrm{dx}}\mathbf{s}(x) = \lim_{\delta x \to 0}\frac{\mathbf{s}(x + \delta x) - \mathbf{s}(x)}{\delta x}$$

The derivative of a dual number is another dual number. Remarkably, the dual operator's condition $\varepsilon^2 = 0$ enables us to take advantage of Taylor series to find the differentiable. Where we can see below, if we substituting a dual number into Taylor series, we get:

$$\begin{aligned}f(r_A + d_A\varepsilon) &= f(r_A) + \frac{f'(r_A)}{1!}d_A\varepsilon + \frac{f''(r_A)}{2!}(d_A\varepsilon)^2 + \frac{f'''(r_A)}{3!}(d_A\varepsilon)^3 + \ldots \\ &= f(r_A) + \frac{f'(r_A)}{1!}d_A\varepsilon + 0 + 0 + \ldots \qquad (as,\ \varepsilon^2 = 0) \\ &= f(r_A) + f'(r_A)d_A\varepsilon\end{aligned}$$

Remarkably, the Taylor series result gives us an exceptionally tidy answer; from this we use dual number arithmetic and substitution to find the solution to any differential.

The derivative also enables us to find the tangent of an arbitrary point p on a given parametric curve that is equal to the normalized dual part of the point p.

6. QUATERNIONS

Quaternions were introduced by Hamilton in 1866 [HAMI86] and have had a rollercoaster of a time with acceptance. Quaternions are an extension of complex number-theory to formulate a four dimensional manifold. A quaternion is defined as:

$$\mathbf{q} = w + (x\mathbf{i} + y\mathbf{j} + z\mathbf{k})$$

where *w*, *x*, *y* and *z* are the numerical values, while *i*, *j* and *k* are the imaginary components.

The imaginary components properties:

$$i^2 = j^2 = k^2 = -1$$

and

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

It is more common to represent the quaternion as two components, the vector component (x, y and z) and the scalar component (w).

$$\mathbf{q} = (w, \mathbf{v})$$

For further reading on the workings of quaternions and their advantages I highly recommend reading McDonalds [MCDO10] introductory paper for students.

6.1. Quaternion Arithmetic Operations

Since we are combining quaternions with dual number theory, we give the elementary quaternion arithmetic operations below:

Scalar Multiplication

$$s\mathbf{q} = (sw, s\mathbf{v})$$

where *s* is a scalar value.

Addition

$$\mathbf{q}_1 + \mathbf{q}_2 = (w_1 + w_2,\ \mathbf{v}_1 + \mathbf{v}_2)$$

Multiplication

$$\mathbf{q}_1\mathbf{q}_2 = (w_1w_2 - v_1v_2,\ \ w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + (\mathbf{v}_1 \times \mathbf{v}_2))$$

Conjugate

$$\mathbf{q}^* = (w, -\mathbf{v})$$

Magnitude

$$\| \mathbf{q} \| = \mathbf{q}\mathbf{q}^*$$

For a unit quaternion, $\| \mathbf{q} \| = 1$. The unit quaternion is used to represent a rotation of an angle θ, radians about a unit axis $\mathbf{n}$, in three-dimensional space:

$$\mathbf{q} = (\cos(\frac{\theta}{2}),\ \mathbf{n}\sin(\frac{\theta}{2}))$$

6.2. Quaternion Interpolation

An extremely important quality of quaternions that make them indispensable in animation systems is their ability to interpolate two or more quaternions smoothly and continuously. Shoemake [SHOE85], presents an outstanding paper on using quaternion curves for animating rotations. Furthermore, it should be noted, the spherical linear interpolation (SLERP) properties of quaternions are inherited by dual-quaternions.

7. DUAL-QUATERNIONS

When quaternions are combined with dual number theory, we get dual-quaternions which was presented by Clifford in 1882 [CLIF82]. While the unit quaternion only has the ability to represent rotation, the unit dual-quaternion can represent both translation and rotation. Each dual-quaternion consists of eight elements or two quaternions. The two quaternion elements are called the real part and the dual part.

$$\mathbf{q} = \mathbf{q}_r + \mathbf{q}_d\varepsilon$$

where $\mathbf{q}_r$ and $\mathbf{q}_d$ are quaternions. Combining the algebra operations associated with quaternions with the additional dual number ε, we can form the dual-quaternion arithmetic.

7.1. Dual-Quaternion Arithmetic Operations

The elementary arithmetic operations necessary for us to use dual-quaternions are:

Scalar Multiplication

$$s\mathbf{q} = s\mathbf{q}_r + s\mathbf{q}_d\varepsilon$$

Addition

$$\mathbf{q}_1 + \mathbf{q}_2 = \mathbf{q}_{r1} + \mathbf{q}_{r2} + (\mathbf{q}_{d1} + \mathbf{q}_{d2})\varepsilon$$

Multiplication

$$\mathbf{q}_1\mathbf{q}_2 = \mathbf{q}_{r1}\mathbf{q}_{r2} + (\mathbf{q}_{r1}\mathbf{q}_{d2} + \mathbf{q}_{d1}\mathbf{q}_{r2})\varepsilon$$

Conjugate

$$\mathbf{q}^* = \mathbf{q}_r^* + \mathbf{q}_d^*\varepsilon$$

Magnitude

$$\| \mathbf{q} \| = \mathbf{q}\mathbf{q}^*$$

Unit Condition

$$\| \mathbf{q} \| = 1$$

$$\mathbf{q}_r^*\mathbf{q}_d + \mathbf{q}_d^*\mathbf{q}_r = 0$$

The unit dual-quaternion is our key concern as it can represent any rigid rotational and translational transformations.

The rigid rotational and translational information for the unit dual-quaternion is:

$$\mathbf{q}_r = \mathbf{r}$$
$$\mathbf{q}_d = \frac{1}{2}\,\mathbf{t}\,\mathbf{r}$$

where $\mathbf{r}$ is a unit quaternion representing the rotation and $\mathbf{t}$ is the quaternion describing the translation represented by the vector $\mathbf{t} = (0, \vec{v})$.

The dual-quaternion can represent a pure rotation the same as a quaternion by setting the dual part to zero.

$$\mathbf{q}_r = [\cos(\frac{\theta}{2}),\ \mathbf{n}_x \sin(\frac{\theta}{2}),\ \mathbf{n}_y \sin(\frac{\theta}{2}),\ \mathbf{n}_z \sin(\frac{\theta}{2})][0,0,0,0]$$

To represent a pure translation with no rotation, the real part can be set to an identity and the dual part represents the translation.

$$\mathbf{q}_t = [1,0,0,0][0, \frac{\mathbf{t}_x}{2}, \frac{\mathbf{t}_y}{2}, \frac{\mathbf{t}_z}{2}]$$

Combining the rotational and translational transforms into a single unit quaternion to represent a rotation followed by a translation we get:

$$\mathbf{q} = \mathbf{q}_t \times \mathbf{q}_r$$

This arithmetic operation defines how we transform a point *p*, using a unit dual-quaternion:

$$\mathbf{p}' = \mathbf{q}\,\mathbf{p}\,\mathbf{q}^*$$

where $\mathbf{q}$ and $\mathbf{q}^*$ represent a dual-quaternion transform and its conjugate; while $\mathbf{p}$ and $\mathbf{p}'$ represent our point inserted into a quaternion and its resulting transform.

8. PORTING EXISTING CODE TO DUAL-QUATERNIONS

A dual-quaternion consists of two quaternions, but is represented by a single variable Q. Systems that have been constructed using separate translation and rotation (vector for translation and quaternion for rotation) in combination with matrices schemes are easily modified to use dual-quaternions for spatial information.

1. For each link, construct a dual-quaternion Q from the rotation and translation information.
2. Real part of the quaternion is the rotation quaternion *r*. The dual part is calculated by multiplying the quaternion *r* and translation component *t*, e.g.:

$$Qr = r$$
$$Qd = 0.5\ (0,t)\ r$$

3. Combine transformations as you would matrices using multiplication.
4. If necessary, for long chains, the dual-quaternion should be re-normalized (to mend drift and maintain a unit dual-quaternion).
5. To get the homogeneous transformation matrix, convert the dual-quaternion by extracting the translational and rotational components.
6. The extracted rotation quaternion r and vector translation information is extracted using:

$$r = Qr$$
$$t = 2\ Qd\ Qr^*$$

Dual-quaternion multiplication is more efficient than matrix multiplication and can effortlessly be converted back to a matrix when needed. Dual-quaternions, unlike Euler angles, do not present issues like "*gimbal lock*" and hence, are ideal for complex articulated hierarchies.

9. COMPLEX CHARACTER HIERARCHY FORWARD KINEMATICS

The focus of our attention is with rigid hierarchies having a large number of DOF. Humans have a tremendous amount of flexibility which we emulate and analyze using numerical and mathematical models. Forward kinematics is the method of concatenating local positions and rotations together to give their global ones. The forward kinematic method for concatenating transforms is the same for dual-quaternions and matrices; which use simple multiplication to propagate transforms between the connected links.

For example, the concatenation of transforms with matrices and dual-quaternions:

Matrix

$$\mathbf{M}_{03} = \mathbf{M}_0\mathbf{M}_1\mathbf{M}_2\mathbf{M}_3$$

Dual-Quaternion

$$\mathbf{q}_{03} = \mathbf{q}_0\mathbf{q}_1\mathbf{q}_2\mathbf{q}_3$$

where the subscript represents the transform, matrix transform $\mathbf{M}_0$ corresponding to dual-quaternion transform $\mathbf{q}_0$.

10. EXPERIMENTAL RESULTS

We used traditional matrix methods during initial character transformation experiments; e.g., inverse kinematic (IK) and animation blending to demonstrate their numerous problems. Matrix methods are a popular choice and solutions to these problems have been developed; we used some of these engineering solutions. Of course, these workarounds to these problems introduced an additional computational cost. Furthermore, certain circumventions to overcome a problem often introduced errors in other areas. One such engineering solution for reducing the impact of drift

and concatenation error was to renormalize the matrices at each level (and at each update frame). The error reduced skewing and scaling but manifested itself in the ideal global end-link orientations and positions being inaccurate.

To demonstrate the problems, we constructed numerous test cases to emphasis them. We also demonstrate and explore how dual-quaternions can represent rigid body character based systems.

10.1. Rigid Body Transform Chains

We constructed a straightforward IK solver that would follow a target end-effector. To mimic how a character would move his arm or leg. The end-effector had six DOF, which the IK solver had to work with to meet its target goal.

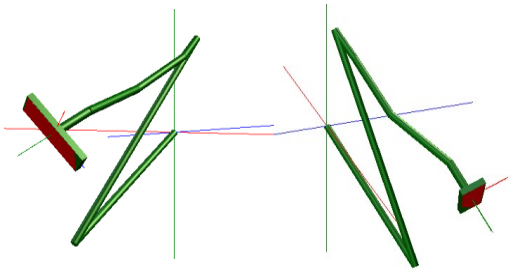

Figure 2. Rigid body links attached in a single hieararchy frame. (Draw ideal(red) and calculated end-effector (green).

The hierarchy is composed of rigid links. Each link held a rotation and translation in the form of a matrix or dual-quaternion. For calculations, the axis-angle and translation could be extracted and used when needed. Local transforms were combined from the root to the end-effectors. Concatenation of the transforms throughout the levels was achieved by multiplying parent transforms with current transforms.

Certain orientation and translation configurations produced errors in the output, shown in Figure 3. These errors presented themselves as skewing and scaling manifestations.

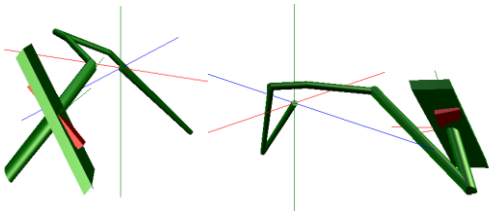

Figure 3. Artifact error when matrices representing translation and orientation in linked hierarchies.

Early workarounds to amend the problem were to repair the matrix at each level in the hierarchy by ortho-normalizing the rotational component. While ortho-normalizing the matrix reduced scaling and skewing artifacts, alternative errors manifest themselves in alternative forms.

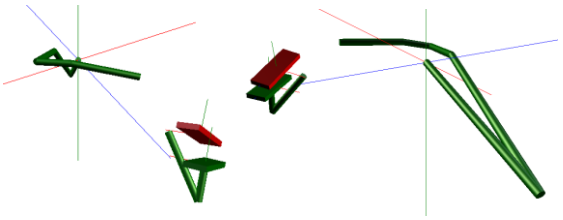

Figure 4. Ortho-normalizing matrices in hierarchies in an attempt to reduce errors.

Ortho-normalizing the rotational part of the transform matrix between updates removed scaling and skewing problems. The joints presented discontinuity errors in the frames hierarchy (see Figure 4). The ideal end-effectors position and rotation were also different from the calculated one using the refurbish matrices.

10.2. Biped Model

For our test character, we used a 16 link biped model, shown in Figure 5. The character has 36 degrees of freedom (DOF). Character rigs can produce extremely non-linear motions due to their joint limits, flexibility and elaborate arrangement of joints.

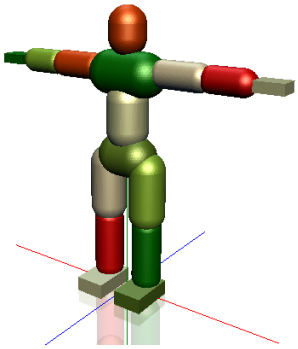

Figure 5. 16 link biped model used for testing.

Figure 5, shows the biped model in its starting stance pose.

Buildup of computational inaccuracies will cause a dual-quaternion to become of non-unit length; we fix these errors by renormalization. In contrast, repairing a non-orthogonal matrix is much more complicated (see [SALA79]).

11. RESULTS

The dual-quaternion unifies the translation and rotation into a single state variable. This single state variable offers a robust, unambiguous, computationally efficient way of representing rigid transform.

The computational cost of combining matrices and dual-quaternions:

Matrix4x4	: 64mult + 48adds
Matrix4x3	: 48mult + 32adds
DualQuaternion	: 42mult + 38adds

In our tests, we found the dual-quaternion multiplication method of transforms on average ten percent faster compared matrix multiplication. We did not take advantage of CPU architecture using parallel methods such as SIMD which can further improve speeds as demonstrated by Pallavi [MEHU10] (both for matrices and quaternion multiplication).

One major advantage we found when working with dual-quaternions was the added advantage of calculating angular and linear differences. When working with pure matrix methods we needed to convert the matrix to a quaternion to calculate angular variations.

12. CONCLUSION AND FURTHER WORK

The dual-quaternion model is an accurate, computationally efficient, robust, and flexible method of representing rigid transforms and should not be overlooked. Implementing pre-programmed dual-quaternion modules (e.g., multiplication and normalization) enables the creation of more elegant and clearer computer programs that are easier to work with and control.

While matrices are the *de-facto* method used for the majority of hierarchy based simulations, we have shown that they can present certain problems which are costly to avoid (e.g., renormalizing a matrix). The problem and cost of drifting and normalizing is less with dual-quaternions compared to matrix methods. When dealing with rigid transforms the dual-quaternion method shines through due to its numerous advantages.

This paper has only provided a taste of the potential advantages of dual-quaternions, and one can only imagine the further future possibilities that they can offer. For example, there is a deeper investigation of the mathematical properties of dual-quaternions (e.g., zero divisions). There is also the concept of dual-dual-quaternions (i.e., dual numbers within dual numbers) and calculus for multi-parametric objects for the reader to pursue if he desires.

13. APPENDIX

13.1. Dual-Quaternion Implementation Class.

```
public class DualQuaternion_c
{
public Quaternion   m_real;
public Quaternion   m_dual;
public DualQuaternion_c()
{
  m_real = new Quaternion(0,0,0,1);
  m_dual = new Quaternion(0,0,0,0);
}
public DualQuaternion_c( Quaternion r, Quaternion d )
{
  m_real = Quaternion.Normalize( r );
  m_dual = d;
}
public DualQuaternion_c( Quaternion r, Vector3 t )
{
  m_real   = Quaternion.Normalize( r );
  m_dual  = ( new Quaternion( t, 0 ) * m_real ) * 0.5f;
}
public    static    float    Dot(    DualQuaternion_c    a,
DualQuaternion_c b )
{
  return Quaternion.Dot( a.m_real, b.m_real );
}
public static DualQuaternion_c operator* (DualQuaternion_c
q, float scale)
{
  DualQuaternion_c ret = q;
  ret.m_real *= scale;
  ret.m_dual *= scale;
  return ret;
}
public static DualQuaternion_c Normalize( DualQuaternion_c q
)
{
  float mag = Quaternion.Dot( q.m_real, q.m_real );
  Debug_c.Assert( mag > 0.000001f );
  DualQuaternion_c ret = q;
  ret.m_real *= 1.0f / mag;
  ret.m_dual *= 1.0f / mag;
  return ret;
}
public static DualQuaternion_c operator + (DualQuaternion_c
lhs, DualQuaternion_c rhs)
{
  return new DualQuaternion_c(lhs.m_real + rhs.m_real,
    lhs.m_dual + rhs.m_dual);
}
// Multiplication order - left to right
public static DualQuaternion_c operator * (DualQuaternion_c
lhs, DualQuaternion_c rhs)
{
  return new DualQuaternion_c(rhs.m_real*lhs.m_real,
     rhs.m_dual*lhs.m_real + rhs.m_real*lhs.m_dual);
}
public static DualQuaternion_c Conjugate( DualQuaternion_c q
)
{
  return    new    DualQuaternion_c(    Quaternion.Conjugate(
q.m_real ), Quaternion.Conjugate( q.m_dual ) );
}
public static Quaternion GetRotation( DualQuaternion_c q )
{
  return q.m_real;
}
public static Vector3 GetTranslation( DualQuaternion_c q )
{
  Quaternion t = ( q.m_dual * 2.0f ) * Quaternion.Conjugate(
q.m_real );
  return new Vector3( t.X, t.Y, t.Z );
}
public      static      Matrix      DualQuaternionToMatrix(
DualQuaternion_c q )
{
  q = DualQuaternion_c.Normalize( q );

  Matrix M = Matrix.Identity;
  float w = q.m_real.W;
  float x = q.m_real.X;
  float y = q.m_real.Y;
  float z = q.m_real.Z;

  // Extract rotational information
  M.M11 = w*w + x*x - y*y - z*z;
  M.M12 = 2*x*y + 2*w*z;
  M.M13 = 2*x*z - 2*w*y;

  M.M21 = 2*x*y - 2*w*z;
  M.M22 = w*w + y*y - x*x - z*z;
  M.M23 = 2*y*z + 2*w*x;

  M.M31 = 2*x*z + 2*w*y;
  M.M32 = 2*y*z - 2*w*x;
  M.M33 = w*w + z*z - x*x - y*y;

  // Extract translation information
  Quaternion t = (q.m_dual * 2.0f) * Quaternion.Conjugate(
q.m_real);
  M.M41 = t.X;
  M.M42 = t.Y;
  M.M43 = t.Z;
  return M;
```

```
}

#if false
public static void SimpleTest()
{
  DualQuaternion_c     dq0     =     new     DualQuaternion_c(
Quaternion.CreateFromYawPitchRoll(1,2,3),                new
Vector3(10,30,90) );
  DualQuaternion_c     dq1     =     new     DualQuaternion_c(
Quaternion.CreateFromYawPitchRoll(-1,3,2),               new
Vector3(30,40, 190 ) );
  DualQuaternion_c     dq2     =     new     DualQuaternion_c(
Quaternion.CreateFromYawPitchRoll(2,3,1.5f),             new
Vector3(5,20, 66 ) );
  DualQuaternion_c dq = dq0 * dq1 * dq2;


  Matrix                    dqToMatrix                    =
DualQuaternion_c.DualQuaternionToMatrix( dq );

  Matrix   m0   =   Matrix.CreateFromYawPitchRoll(1,2,3)   *
Matrix.CreateTranslation( 10, 30, 90 );
  Matrix   m1   =   Matrix.CreateFromYawPitchRoll(-1,3,2)   *
Matrix.CreateTranslation( 30, 40, 190 );
  Matrix   m2   =   Matrix.CreateFromYawPitchRoll(2,3,1.5f)   *
Matrix.CreateTranslation( 5, 20, 66 );
  Matrix m = m0 * m1 * m2;
}
#endif
} // End DualQuaternion_c
```

13.2. Novice Errors

There are a few things to look out for when implementing a dual-quaternion class. Firstly, ensure the multiplication order is correct and remains consistent with matrices (i.e., left to right). Secondly, always ensure that the dual-quaternions remain normalized (i.e., unit-length).

14. REFERENCES

[CLIF82] W. Clifford, Mathematical Papers. London: Macmillan, 1882.

[KCŽO08] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan, "Geometric skinning with approximate dual quaternion blending," ACM Transactions on Graphics (TOG), vol. 27, no. 4, p. 105, 2008.

[IVIV11] F. Z. Ivo and H. Ivo, "Spherical skinning with dual quaternions and QTangents," ACM SIGGRAPH 2011 Talks, vol. 27, p. 4503, 2011.

[SELI11] J. Selig, "Rational interpolation of rigid-body motions," Advances in the Theory of Control, Signals and Systems with Physical Modeling, pp. 213–224, 2011.

[VAFU09] A. Vasilakis and I. Fudos, "Skeleton-based rigid skinning for character animation," in Proc. of the Fourth International Conference on Computer Graphics Theory and Applications, 2009, no. February, pp. 302–308.

[KMLX11] Y. Kuang, A. Mao, G. Li, and Y. Xiong, "A strategy of real-time animation of clothed body movement," in Multimedia Technology (ICMT), 2011 International Conference on, 2011, pp. 4793–4797.

[PPAF10] H. L. Pham, V. Perdereau, B. V. Adorno, and P. Fraisse, "Position and orientation control of robot manipulators using dual quaternion feedback," in Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, 2010, pp. 658–663.

[SCHI11] M. Schilling, "Universally manipulable body models — dual quaternion representations in layered and dynamic MMCs," Autonomous Robots, 2011.

[GVMC98] Q. Ge, A. Varshney, J. P. Menon, and C. F. Chang, "Double quaternions for motion interpolation," in Proceedings of the ASME Design Engineering Technical Conference, 1998.

[LIWC10] Y. Lin, H. Wang, and Y. Chiang, "Estimation of relative orientation using dual quaternion," System Science and, no. 2, pp. 413-416, 2010.

[PEMC04] A. Perez and J. M. McCarthy, "Dual quaternion synthesis of constrained robotic systems," Journal of Mechanical Design, vol. 126, p. 425, 2004.

[ALMA92] W. Alan and W. Mart, Advanced Animation and Rendering Techniques: Theory and Practice. Adison-Wesley, 1992.

[GILB86] S. Gilbert, Introduction to Applied Mathematics. Wellesley-Cambridge Press, 1986.

[BERG09] G. van den Bergen, "Dual Numbers: Simple Math, Easy C++ Coding, and Lots of Tricks," GDC Europe, 2009. [Online]. Available: www.gdcvault.com/play/10103/Dual-Numbers-Simple-Math-Easy.

[HAMI86] W. R. Hamilton, Elements of Quaternions. London: , 1886.

[MCDO10] J. McDonald, "Teaching Quaternions is not Complex," Computer Graphics Forum, vol. 29, no. 8, pp. 2447-2455, Dec. 2010.

[SHOE85] K. Shoemake, "Animating rotation with quaternion curves," ACM SIGGRAPH computer graphics, 1985.

[SALA79] E. Salamin, "Application of quaternions to computation with rotations," Internal Report, Stanford University, Stanford, CA, vol. 1, 1979.

[MEHU10] P. Mehrotra and R. Hubbard, "Benefits of Intel® Advanced Vector Extensions For Quaternion Spherical Linear Interpolation (Slerp)," 2010. [Online]. Available: http://software.intel.com/en-us/articles/benefits-of-intel-advanced-vector-extensions-for-quaternion-spherical-liner-interpolation-slerp/.