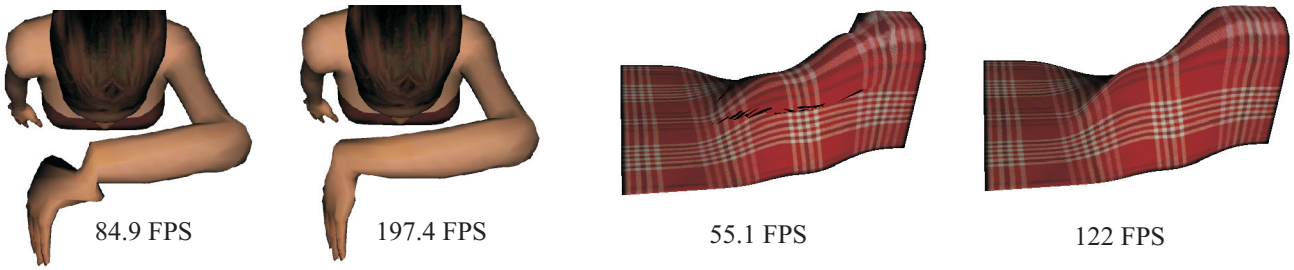
Dual Quaternions for Rigid Transformation Blending

Ladislav Kavan*
Trinity College Dublin / CTU in Prague

Steven Collins
Trinity College Dublin

Carol O'Sullivan
Trinity College Dublin

Jiri Zara
CTU in Prague



(a) Log-matrix Blending **(b)** Dual Quaternions **(c)** Spherical Blend Skinning **(d)** Dual Quaternions

Figure 1: A comparison of different rigid transformation blending algorithms applied in skinning shows that blending based on dual quaternions not only eliminates artifacts, but is also much easier to implement and more than twice as fast as previous methods.

Abstract

Quaternions have been a popular tool in 3D computer graphics for more than 20 years. However, classical quaternions are restricted to the representation of rotations, whereas in graphical applications we typically work with rotation composed with translation (i.e., a rigid transformation). Dual quaternions represent rigid transformations in the same way as classical quaternions represent rotations. In this paper we show how to generalize established techniques for blending of rotations to include all rigid transformations. Algorithms based on dual quaternions are computationally more efficient than previous solutions and have better properties (constant speed, shortest path and coordinate invariance). For the specific example of skinning, we demonstrate that problems which required considerable research effort recently are trivial to solve using our dual quaternion formulation. However, skinning is only one application of dual quaternions, so several further promising research directions are suggested in the paper.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling – Geometric Transformations—[I.3.7]: Computer Graphics—Three-Dimensional Graphics and Realism – Animation

Keywords: rigid transformation, dual quaternion, transformation blending, rigid body motion

1 Introduction

It is well known that classical quaternions are an advantageous representation of 3D rotations, in many aspects better than 3×3 rotation matrices [Shoemake 1985]. However, rigid objects do not only rotate, but also translate; a rotation composed with translation is called a *rigid transformation*. Any displacement of a rigid object in 3D space can be described by a rigid transformation (known therefore also as a rigid body motion). In this paper, we advocate that dual quaternions [Clifford 1882] are in many aspects a better representation of rigid transformations than those treating rotation and translation components independently, such as 4×4 homogeneous matrices, or pairs consisting of a classical quaternion and a translation vector.

We consider the problem of weighted averages, or *blending*, of 3D rigid transformations. The input consists of n rigid transformations and n weights $w_1, \ldots, w_n$ which are convex (i.e., $w_i \geq 0$, and $w_1 + \ldots + w_n = 1$). The task is to compute a weighted average of these transformations. A similar problem is transformation interpolation: the input is also n rigid transformations and n key times, but the task is to construct an interpolation curve. Both problems are equivalent for $n = 2$, but for $n > 2$, blending is more general. This is because any interpolation (e.g., linear, spline) can be expressed as a blending with weights considered as functions of time: $w_1(t), \ldots, w_n(t)$ (see [Buss and Fillmore 2001]). Even though transformation interpolation is very important in computer graphics, there are many applications where the more general blending is required instead: e.g., character skinning, motion blending, spatial keyframing and animation compression [Alexa 2002].

Transformation blending is well studied for 3D rotations. In the case of only two rotations, the established solution is Spherical Linear Interpolation (SLERP), introduced by Shoemake [1985]. For cases involving more than two 3D rotations, there are several solutions possible, e.g., an iterative solution based on spherical averages [Buss and Fillmore 2001], or fast approximate Quaternion Linear Blending (QLB) [Kavan and Zara 2005] (called Quaternion Linear Interpolation (QLERP) in their paper).

For blending of general 3D rigid transformations, there are no proven algorithms described in the literature. In the past, this problem has typically been solved by ad-hoc algorithms, which work correctly only in the context of a specific application. In this paper, we show how dual quaternions can be used to generalize the well-established rotation blending algorithms to deal with all rigid transformations. We present three algorithms:

ScLERP (Screw Linear Interpolation), a generalization of SLERP [Shoemake 1985]

DLB (Dual quaternion Linear Blending), a generalization of QLB [Kavan and Zara 2005]

DIB (Dual quaternion Iterative Blending), a generalization of spherical averages [Buss and Fillmore 2001]

*e-mail: kavanl1@fel.cvut.cz

We prove that all presented algorithms have the same desirable features as their rotation-only counterparts. If we employ dual quaternions in applications, we obtain more accurate results and in many cases also faster execution time. Some tasks that have challenged researchers in recent years are trivial to solve using a dual quaternion approach. We demonstrate this for the problem of geometric skinning, which is an algorithm in common use for real-time animation of virtual characters. The implementation of a dual quaternion class is very simple, especially when based on existing code for classical quaternions. We believe that dual quaternions will become a standard part of every 3D graphics library, in the same way as the original quaternions are today.

Our Contribution. Previous texts present dual quaternions typically in a theoretical, algebraic fashion e.g., [Bottema and Roth 1979]. In Section 3, we introduce the theory of dual quaternions from a practical point of view, emphasizing its application in computer graphics. Rigid transformation blending based on dual quaternions is, to our knowledge, novel. So too is our verification of the properties of associated algorithms (Section 4 and Appendix A). A rigorous comparison with different blending algorithms has been undertaken (Section 5) and a case study, where our new methods are applied to skinning, is also presented (Section 6).

2 Background and Related Work

Dual quaternions were developed by Clifford in the nineteenth century [Clifford 1882]. They are a special case of the structures known today as Clifford algebras, which embrace complex numbers, quaternions, dual quaternions, and others [McCarthy 1990]. Clifford algebras are sometimes called geometric algebras, whose importance for computer graphics has been discussed recently [Hildenbrand et al. 2004]. Very few applications of dual quaternions in computer graphics have been described to date: construction of interpolation curves [Juttler 1994; Ge and Ravani 1994] and inverse kinematics [Luciano and Banerjee 2000]. This suggests that dual quaternions are virtually unknown in the computer graphics community. This is surprising, because they are quite popular in other fields, such as robotics [Daniilidis 1999; Perez and McCarthy 2004].

2.1 Two transformations

Before we address the more complex problem of rigid transformation blending, let us examine first the simple case of $n = 2$, where blending is equivalent to interpolation. Let M_0 and M_1 be two rigid transformations (represented by 4×4 homogeneous matrices), and $t \in [0,1]$ the interpolation parameter. Any interpolation of two rigid transformations, denoted as $\Phi(t, M_0, M_1)$, is considered valid if it satisfies the following basic requirements: $\Phi(0, M_0, M_1) = M_0$, $\Phi(1, M_0, M_1) = M_1$ and $\Phi(t, M_0, M_1)$ is a rigid transformation for all $t \in [0,1]$. Moreover, the interpolation should be independent of the order of input transformations, i.e., $\Phi(t, M_0, M_1) = \Phi(1-t, M_1, M_0)$. However, even when taking all these requirements into account, there are still an infinite number of ways to define a valid interpolation. It is thus desirable to pick an interpolation method which satisfies the most advantageous additional properties.

For the case of two 3D rotations R_0, R_1, the popular Spherical Linear Interpolation (SLERP) algorithm [Shoemake 1985] has the following properties:

- *Constant speed* means that the angle of interpolated rotation varies linearly with respect to parameter t.
- *Shortest path* means that the motion between R_0 and R_1 is a rotation about a fixed axis with the smallest angle.
- *Coordinate system invariance* requires that it does not matter whether a conversion to any other arbitrary coordinate system occurs before or after interpolation.

A formal definition of those properties (for general rigid transformations) is provided below. The constant speed, shortest path and coordinate invariance properties are the reason why SLERP is superior to other methods. Let us therefore examine those properties in detail. The importance of coordinate invariance is very well-known in geometry and robotics [Murray et al. 1994]. In computer graphics papers concerning transformation blending, coordinate-invariance is usually not discussed. This is surprising, because almost all transformation blending algorithms actually are coordinate-invariant, see Section 5. We presume this is because the concept of coordinates is ingrained in computer graphics, e.g., every application works in some "world-coordinate system", all modeling software displays coordinate axes, etc. The importance of coordinate-invariance in computer animation has been emphasized by Lee and Shin [2002].

The matrix representation of a geometric transformation depends inherently on the chosen coordinate system. We obtain a different matrix representation for a different choice of coordinate system, even though the transformation is still the same. Ideally, we do not want the blending method to depend on the *representation* of the transformation, as we prefer only dependence on the *transformation itself*. This is the meaning of coordinate-invariance. A practical argument is that certain algorithms do not work with coordinate-dependent blending [Ge et al. 1998; Kavan and Zara 2005]. However, we do not claim that a coordinate-invariant algorithm is the best solution in all situations: for example, if we are just interpolating the position of a teapot (e.g., as in Figure 2) it would be more natural to rotate the teapot around its center of mass, rather than around the screw axis. However, this reasoning assumes that we know beforehand that 1) we are transforming a teapot, and 2) we know its mass distribution (or at least a user-defined approximation of its center of mass). Such a priori information may not be available in all situations, or it may be too time consuming to compute (e.g., center of mass of a deformable object). A coordinate invariant algorithm is an elegant solution in this case: it works equally for all coordinate systems, thus it does not matter which one we use. It is interesting to note that coordinate-invariance is also the reason why the weights $w_1, \ldots, w_n$ of a linear combination of points are required to satisfy the equation $w_1 + \ldots + w_n = 1$. It would of course be possible to compute a linear combination of points with weights not summing to one, but the result would be dependent on the coordinate system with respect to which the points are expressed.

Constant speed and shortest path properties are more intuitive: they correspond to the motion with the smallest amount of work (any acceleration or motion of the rotation axis implies additional energy). This is why the blending looks natural: if we were to perform the motion ourselves, we would also tend to avoid unnecessary work.

Formal definition of interpolation properties. Let $\Phi(t, M_0, M_1)$ denote a valid interpolation between rigid transformations M_0 and M_1 with parameter $t \in [0,1]$. The rigid transformation $M_0^{-1}\Phi(t, M_0, M_1)$ can be decomposed to angle of rotation $\alpha(t)$, unit axis of rotation $\mathbf{a}(t)$, amount of translation $\delta(t)$ and unit translation direction $\mathbf{d}(t)$. The interpolation is called

- *Constant speed* if the derivative of both $\alpha(t)$ and $\delta(t)$ is constant.

• *Shortest path* if $\mathbf{a}(t)$ and $\mathbf{d}(t)$ are constant, and $\alpha(1) \in [-\pi, \pi]$.

• *Coordinate system invariant* if, for an arbitrary rigid transformation T, is true that $T\Phi(t, M_0, M_1)T^{-1} = \Phi(t, TM_0T^{-1}, TM_1T^{-1})$.

2.2 Multiple transformations

A limitation of SLERP is that it cannot be directly generalized for the blending of more than two rotations. For example, the obvious generalization to three rotations R_0, R_1, R_2 is to first compute SLERP of R_0 with R_1 and then interpolate the result with R_2. This solution has a serious drawback, which is the dependence on the order of rotations: if we reorder the rotations e.g., to R_2, R_1, R_0 (and change the blending weights accordingly), we obtain a different result. A better generalization of SLERP, independent of the order of rotations, has been described in [Buss and Fillmore 2001]. A similar algorithm has been also discussed by Johnson [2003]. However, both of the above algorithms are iterative and slower than the closed-form SLERP. For applications in real-time graphics, a fast closed-form approximation has been advocated [Kavan and Zara 2005]. It works by using a simple linear combination of unit quaternions followed by projection onto a unit hypersphere. These methods are however limited only to rotations. For the case of general rigid transformations, the following algorithms have been used in the past:

• **Linear blending of matrices.** The component-wise linear blending of rigid transformation matrices is the most straightforward solution. The problem with this method is that its result is often not a rigid transformation matrix. The blended matrix can even be singular (projecting 3D objects to 2D).

• **Decomposition to rotation and translation.** An obvious solution to the problems of linear blending of matrices is to decompose the rigid transformation matrix into a rotation and translation component. The translation part can be blended linearly without any difficulties, and the rotation part can be converted to a quaternion and blended properly using [Buss and Fillmore 2001]. However, there is an associated hidden drawback: the decomposition to rotation and translation inevitably depends on the coordinate system, thus we lose the convenient property of coordinate independence. If this is to work reasonably, we must provide a suitable coordinate system in which the decomposition will be performed.

The construction of a such a coordinate system has been presented in [Kavan and Zara 2005]. Unfortunately, the proposed computation of a coordinate system is based on a least squares optimization, solved by the Singular Value Decomposition (SVD) algorithm. This is restrictive, because SVD is a complex and slow algorithm, unsuitable for real-time applications. Moreover, the computation of the auxiliary coordinate system does not depend on the blending weights – it is thus doubtful whether this can really be considered as a correct blending. Note that no such problems occur with coordinate-invariant blending.

• **Blending of matrix logarithms.** The advantage of the transformation blending algorithm based on matrix logarithms [Alexa 2002] is that it works with more general transformations (not just rigid). However, the manner in which it blends rotations has opened an interesting discussion. An on-line critique of Alexa's article [Bloom et al. 2004] was presented, which points out that Alexa's blending applied for rotations is not shortest path, i.e., it can produce a longer trajectory than necessary. However, Bloom et al. do not propose any alternative solution for the case of general transformations. In this paper, we show how Alexa's original algorithm for general transformations can be upgraded to be both shortest path and constant speed. However, for the case of rigid transformations, we find our proposed algorithms based on dual quaternions to be much more practical (in terms of both robustness and computational speed). A compact representation of a rigid transformation matrix logarithm is a twist [Bregler and Malik 1998] or hexanion [Angelidis 2004]. Blending of those entities is equivalent to Alexa's method.

Conventions. We denote scalars by lower-case letters, vectors and quaternions by bold and matrices by capital letters. Dual quantities are distinguished from non-dual by a caret, for example, $\hat{a}$ denotes a dual number and $\hat{\mathbf{q}}$ a dual quaternion. The i-th component of vector $\mathbf{v}$ is written as v_i, thus also $\mathbf{v} = (v_1, \ldots, v_n)$. The dot product of vectors $\mathbf{v}$ and $\mathbf{w}$ is denoted as $\langle \mathbf{v}, \mathbf{w} \rangle$ and the norm $\|\mathbf{v}\|$ is a shortcut for $\sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$. Cross product is denoted as $\mathbf{v} \times \mathbf{w}$. For better readability, we denote the exponential mapping as $\exp(x)$, instead of e^x.

3 Dual Quaternions

This section provides a brief introduction to dual numbers and dual quaternions. We focus only on the main ideas, omitting some lengthy mathematical proofs, which can typically be done by direct computation. For a more detailed introduction, see [Bottema and Roth 1979; McCarthy 1990]. We assume the reader is already familiar with ordinary quaternions, otherwise see for example [Dam et al. 1998; Eberly 2001]. Dual quaternions can be considered as quaternions whose elements are dual numbers. The algebra of dual numbers is similar to complex numbers: any dual number $\hat{a}$ can be written as $\hat{a} = a_0 + \varepsilon a_\varepsilon$, where a_0 is the non-dual part, a_ε the dual part and ε is a *dual unit* satisfying $\varepsilon^2 = 0$. The dual conjugate is analogous to the complex conjugate: $\overline{\hat{a}} = a_0 - \varepsilon a_\varepsilon$. Multiplication of two dual numbers is given as $(a_0 + \varepsilon a_\varepsilon)(b_0 + \varepsilon b_\varepsilon) = a_0 b_0 + \varepsilon(a_0 b_\varepsilon + a_\varepsilon b_0)$. The inverse of a dual number $\hat{a}^{-1}$ is given by $\frac{1}{a_0 + \varepsilon a_\varepsilon} = \frac{1}{a_0} - \varepsilon \frac{a_\varepsilon}{a_0^2}$, as can be immediately verified. The previous expression is defined only when $a_0 \neq 0$. Purely dual numbers, that is dual numbers with $a_0 = 0$, do not have an inverse. This is a fundamental difference from complex numbers, because every non-zero complex number has an inverse. The square root is defined only for dual numbers with a positive non-dual part, and it is computed as $\sqrt{a_0 + \varepsilon a_\varepsilon} = \sqrt{a_0} + \varepsilon \frac{a_\varepsilon}{2\sqrt{a_0}}$.

A dual quaternion $\hat{\mathbf{q}}$ can be written as $\hat{\mathbf{q}} = \hat{w} + i\hat{x} + j\hat{y} + k\hat{z}$, where $\hat{w}$ is the scalar part (dual number), $(\hat{x}, \hat{y}, \hat{z})$ is the vector part (dual vector), and i, j, k are the usual quaternion units. The dual unit ε commutes with quaternion units, for example $i\varepsilon = \varepsilon i$. A dual quaternion can be also considered as an 8-tuple of real numbers, or as the sum of two ordinary quaternions, $\hat{\mathbf{q}} = \mathbf{q}_0 + \varepsilon \mathbf{q}_\varepsilon$. Conjugation of a dual quaternion is defined using classical quaternion conjugation: $\hat{\mathbf{q}}^* = \mathbf{q}_0^* + \varepsilon \mathbf{q}_\varepsilon^*$ The norm of a dual quaternion can be written as $\|\hat{\mathbf{q}}\| = \sqrt{\hat{\mathbf{q}}^* \hat{\mathbf{q}}} = \sqrt{\hat{\mathbf{q}} \hat{\mathbf{q}}^*}$, which expands to

$$\|\hat{\mathbf{q}}\| = \sqrt{\hat{\mathbf{q}}^* \hat{\mathbf{q}}} = \|\mathbf{q}_0\| + \varepsilon \frac{\langle \mathbf{q}_0, \mathbf{q}_\varepsilon \rangle}{\|\mathbf{q}_0\|}$$

The norm satisfies the usual property $\|\hat{\mathbf{p}}\hat{\mathbf{q}}\| = \|\hat{\mathbf{p}}\| \|\hat{\mathbf{q}}\|$. The inverse of a dual quaternion is defined only when $\mathbf{q}_0 \neq \mathbf{0}$. In this case, we have $\hat{\mathbf{q}}^{-1} = \frac{\hat{\mathbf{q}}^*}{\|\hat{\mathbf{q}}\|^2}$. Unit dual quaternions are those satisfying $\|\hat{\mathbf{q}}\| = 1$. According to the previous formula, a dual quaternion $\hat{\mathbf{q}}$ is unit if and only if $\|\mathbf{q}_0\| = 1$ and $\langle \mathbf{q}_0, \mathbf{q}_\varepsilon \rangle = 0$. Note that unit dual quaternions are always invertible (their inverse is just conjugation). We denote the set of unit dual quaternions as $\hat{Q}_1$. Geometrically, $\hat{Q}_1$ is a manifold in 8-dimensional Euclidean space (called an

image-space of dual quaternions [McCarthy 1990]). Just like ordinary quaternions, dual quaternions are also associative, distributive, but not commutative.

As expected, unit dual quaternions naturally represent 3D rotation, when the dual part $\mathbf{q}_\varepsilon = \mathbf{0}$. If we have a 3D vector (v_0, v_1, v_2), we define the associated unit dual quaternion as $\hat{\mathbf{v}} = 1 + \varepsilon(v_0 i + v_1 j + v_2 k)$. The rotation of vector (v_0, v_1, v_2) by a dual quaternion $\hat{\mathbf{q}}$ then can be written as $\hat{\mathbf{q}}\hat{\mathbf{v}}\overline{\hat{\mathbf{q}}}^*$ (where $\overline{\hat{\mathbf{q}}}^*$ denotes both quaternion and dual conjugation). This is obvious, because if $\mathbf{q}_\varepsilon = \mathbf{0}$ then $\hat{\mathbf{q}} = \mathbf{q}_0$ and $\hat{\mathbf{q}}\hat{\mathbf{v}}\overline{\hat{\mathbf{q}}}^*$ simplifies to

$$\mathbf{q}_0(1 + \varepsilon(v_0 i + v_1 j + v_2 k))\mathbf{q}_0^* = 1 + \varepsilon\mathbf{q}_0(v_0 i + v_1 j + v_2 k)\mathbf{q}_0^*$$

where $\mathbf{q}_0(v_0 i + v_1 j + v_2 k)\mathbf{q}_0^*$ is the familiar formula for rotation by an ordinary quaternion.

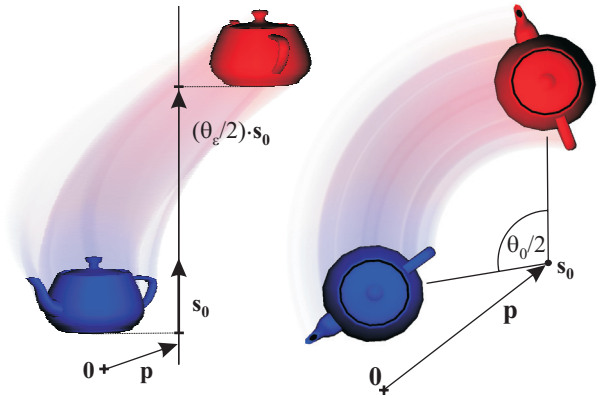


Figure 2: Example of a screw motion, side and top view. Each rigid transformation can be described as a screw: rotation about an axis by angle $\theta_0/2$ and translation with magnitude $\theta_\varepsilon/2$ along the same axis. The axis, with direction determined by unit vector $\mathbf{s}_0$, needs not pass through the origin $\mathbf{0}$. Its position in space is given by vector $\mathbf{p}$ pointing from the origin to some point on the axis. The choice of vector $\mathbf{p}$ is not unique, thus dual quaternions work with the unique moment: $\mathbf{p} \times \mathbf{s}_0$.

What is interesting is that dual quaternions can also represent 3D translation. A unit dual quaternion $\hat{\mathbf{t}}$, defined as $\hat{\mathbf{t}} = 1 + \frac{\varepsilon}{2}(t_0 i + t_1 j + t_2 k)$ corresponds to translation by vector (t_0, t_1, t_2) (note that dual quaternions work with *half* of the translation vector, in analogy to classical quaternions, which work with half of the angle of rotation). If we simplify $\hat{\mathbf{t}}\hat{\mathbf{v}}\overline{\hat{\mathbf{t}}}^*$, we obtain $1 + \varepsilon((v_0 + t_0)i + (v_1 + t_1)j + (v_2 + t_2)k)$, which shows that the unit dual quaternion $\hat{\mathbf{t}}$ really performs translation by (t_0, t_1, t_2). Rigid transformation is a composition of rotation and translation, and composition of transformations corresponds to multiplication of dual quaternions. If the rotation is described by unit quaternion $\mathbf{q}_0$ and the translation by unit dual quaternion $1 + \frac{\varepsilon}{2}(t_0 i + t_1 j + t_2 k)$ as before, then their composition is

$$(1 + \frac{\varepsilon}{2}(t_0 i + t_1 j + t_2 k))\mathbf{q}_0 = \mathbf{q}_0 + \frac{\varepsilon}{2}(t_0 i + t_1 j + t_2 k)\mathbf{q}_0 \qquad (1)$$

We can verify by direct computation that the result is always a unit dual quaternion. Let us assume that we already have a routine for conversion between a 3×3 rotation matrix and a unit quaternion, as well as a routine for quaternion multiplication. Formula (1) then shows how to convert a 4×4 rigid transformation matrix to a unit dual quaternion. The opposite conversion, from a unit dual quaternion $\mathbf{q}_0 + \varepsilon\mathbf{q}_\varepsilon$ to a matrix is also straightforward. The rotation is just a matrix representation of $\mathbf{q}_0$ and the translation is given as $2\mathbf{q}_\varepsilon\mathbf{q}_0^*$.

Every unit dual quaternion $\hat{\mathbf{q}}$ can be written as

$$\hat{\mathbf{q}} = \cos\frac{\hat{\theta}}{2} + \hat{\mathbf{s}}\sin\frac{\hat{\theta}}{2} \qquad (2)$$

where $\hat{\mathbf{s}}$ is a unit dual vector with zero scalar part, see [McCarthy 1990] or [Daniilidis 1999]. Note that this looks like the formula for ordinary quaternions, just employing the dual angle $\hat{\theta} = \theta_0 + \varepsilon\theta_\varepsilon$ and unit dual vector $\hat{\mathbf{s}} = \mathbf{s}_0 + \varepsilon\mathbf{s}_\varepsilon$. The geometric interpretation of those quantities is related to *screw motion*, that is a rotation and translation about the same axis. Chasle's theorem [Daniilidis 1999] states that any rigid transformation can be described by a screw motion, see Figure 2. Angle $\theta_0/2$ is the angle of rotation, and unit vector $\mathbf{s}_0$ represents the direction of the axis of rotation. $\theta_\varepsilon/2$ is the amount of translation along vector $\mathbf{s}_0$, and $\mathbf{s}_\varepsilon$ is the *moment* of the axis. Moment is an unambiguous description of the position of an axis in space. It is given by equation $\mathbf{s}_\varepsilon = \mathbf{p} \times \mathbf{s}_0$, where $\mathbf{p}$ is a vector pointing from the origin to an arbitrary point on the axis. Which point we choose is not important, because for any other point of the axis, say $\mathbf{p} + c\mathbf{s}_0$ (where c is an arbitrary scalar), we obtain the same moment: $(\mathbf{p} + c\mathbf{s}_0) \times \mathbf{s}_0 = \mathbf{p} \times \mathbf{s}_0$. This gives us another insight: whereas classical quaternions can represent only rotations whose axes pass through the origin, dual quaternions can represent rotations with arbitrary axes.

Using Taylor series, we can derive Euler's identity for dual quaternions: $\exp(\hat{\mathbf{s}}\frac{\hat{\theta}}{2}) = \cos\frac{\hat{\theta}}{2} + \hat{\mathbf{s}}\sin\frac{\hat{\theta}}{2}$. If $\hat{\mathbf{q}}$ is as in Formula (2), then its logarithm is given as $log(\hat{\mathbf{q}}) = \hat{\mathbf{s}}\frac{\hat{\theta}}{2}$. A power of dual quaternion $\hat{\mathbf{q}}$ is then defined naturally: $\hat{\mathbf{q}}^{\hat{u}} = \exp(\hat{u}\log\hat{\mathbf{q}}) = \cos(\hat{u}\frac{\hat{\theta}}{2}) + \hat{\mathbf{s}}\sin(\hat{u}\frac{\hat{\theta}}{2})$. We see that the dual quaternion formulas are very similar to corresponding formulas for ordinary quaternions [Dam et al. 1998].

Dual quaternions exhibit the so called *antipodal* property of classical quaternions, i.e., the fact that both $\hat{\mathbf{q}}$ and $-\hat{\mathbf{q}}$ represent the same rigid transformation. The mapping between rigid transformations and unit dual quaternions is thus one to two. Even though both $\mathbf{q}_0$ and $-\mathbf{q}_0$ represent the same rotation, the powers $\mathbf{q}_0^t$ and $(-\mathbf{q}_0)^t$ are different: one corresponds to clockwise and the second to counterclockwise rotation, see Figure 3. During matrix to quaternion conversion, we can choose between $\mathbf{q}_0$ and $-\mathbf{q}_0$. In the following, we assume that the signs corresponding to the shortest trajectory were chosen. In general, this can be done as described in [Park et al. 2002]. In practice, a trivial method is usually possible [Kavan and Zara 2005].

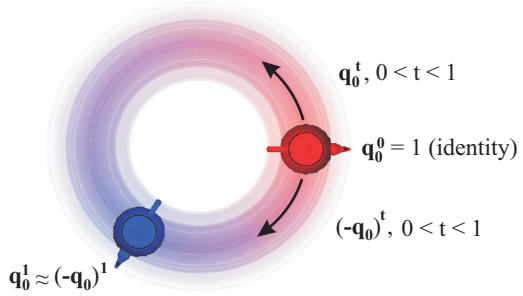


Figure 3: Dual quaternions inherit the antipodality of classical quaternions. In this example, transformation of the teapot by $\mathbf{q}_0^t$ for $t \in [0, 1]$ produces a counterclockwise rotation (longer trajectory), while transformation by $(-\mathbf{q}_0)^t$ leads to a clockwise one (shorter trajectory).

4 Blending of Rigid Transformations

We start by generalizing the famous Spherical Linear Interpolation (SLERP) from rotations to rigid transformations. The interpolation of two unit quaternions $\mathbf{p},\mathbf{q}$ with parameter $t \in [0,1]$ is computed as $SLERP(t;\mathbf{p},\mathbf{q}) = \mathbf{p}(\mathbf{p}^*\mathbf{q})^t$. In the previous section, we have defined the power of dual quaternions, which enables us to straightforwardly generalize SLERP to Screw Linear Interpolation (ScLERP). ScLERP interpolates between two unit dual quaternions $\hat{\mathbf{p}},\hat{\mathbf{q}}$ with parameter $t \in [0,1]$, and is given as $ScLERP(t;\hat{\mathbf{p}},\hat{\mathbf{q}}) = \hat{\mathbf{p}}(\hat{\mathbf{p}}^*\hat{\mathbf{q}})^t$. What is its geometric interpretation? Obviously, $\hat{\mathbf{p}}^*\hat{\mathbf{q}}$ is a unit dual quaternion, which represents a rigid transformation from $\hat{\mathbf{p}}$ to $\hat{\mathbf{q}}$. According to the previous section, the power can be written as $(\hat{\mathbf{p}}^*\hat{\mathbf{q}})^t = \cos(t\frac{\hat{\alpha}}{2}) + \hat{\mathbf{n}}\sin(t\frac{\hat{\alpha}}{2})$ for some dual angle $\hat{\alpha}$ and dual vector $\hat{\mathbf{n}}$. The dual vector $\hat{\mathbf{n}}$ represents the axis of the screw motion (an axis that needs not pass through the origin, as in Figure 2). The dual angle $t\frac{\hat{\alpha}}{2} = t\frac{\alpha_0}{2} + \varepsilon t\frac{\alpha_\varepsilon}{2}$ contains both the angle of rotation ($t\frac{\alpha_0}{2}$) and the amount of translation ($t\frac{\alpha_\varepsilon}{2}$). We can immediately observe two important properties: the axis $\hat{\mathbf{n}}$ of the screw motion is constant (independent of t), and the angle of rotation $t\frac{\alpha_0}{2}$, as well as the amount of translation $t\frac{\alpha_\varepsilon}{2}$, vary linearly with respect to the interpolation parameter t. This means that ScLERP guarantees both shortest path and constant speed interpolation. Lemma 2 in the Appendix A shows that ScLERP is also independent of the choice of a coordinate system. In short, ScLERP is an interpolation for rigid transformations with the same behavior as SLERP for rotations.

In spite of its importance, there are alternatives to SLERP, such as Quaternion Linear Blending (QLB). In the case of two transformations, the inputs of QLB are identical to those of SLERP: quaternions $\mathbf{p},\mathbf{q}$ and parameter $t \in [0,1]$. The formula for QLB is very simple, it is only linear interpolation followed by a normalization, $QLB(t;\mathbf{p},\mathbf{q}) = \frac{(1-t)\mathbf{p}+t\mathbf{q}}{\|(1-t)\mathbf{p}+t\mathbf{q}\|}$. The disadvantage of QLB is that it is not a constant speed interpolation, although it is shortest path [Kavan and Zara 2005], and coordinate-invariant. However, as computed by Kavan and Zara, the difference between the angle of rotation in QLB and SLERP is fairly small, i.e., always strictly less than 8.15 degrees. This means that QLB is actually "almost" constant speed interpolation, but faster and easier to compute than SLERP. Therefore, QLB is usually the method of choice for real-time applications, especially those running on a GPU.

The dual counterpart of QLB is Dual quaternion Linear Blending (DLB). As could be expected, this interpolation is again a direct generalization, $DLB(t;\hat{\mathbf{p}},\hat{\mathbf{q}}) = \frac{(1-t)\hat{\mathbf{p}}+t\hat{\mathbf{q}}}{\|(1-t)\hat{\mathbf{p}}+t\hat{\mathbf{q}}\|}$. The natural question now is whether DLB is also a good approximation of ScLERP, as QLB was of SLERP. We show first the coordinate invariance of DLB. Actually, we show a stronger property, the so called *bi-invariance* [Moakher 2002]. This states that, for any unit dual quaternion $\hat{\mathbf{r}}$, both $DLB(t;\hat{\mathbf{r}}\hat{\mathbf{p}},\hat{\mathbf{r}}\hat{\mathbf{q}}) = \hat{\mathbf{r}}DLB(t;\hat{\mathbf{p}},\hat{\mathbf{q}})$ and $DLB(t;\hat{\mathbf{p}}\hat{\mathbf{r}},\hat{\mathbf{q}}\hat{\mathbf{r}}) = DLB(t;\hat{\mathbf{p}},\hat{\mathbf{q}})\hat{\mathbf{r}}$ (that is, left and right invariance). Bi-invariance implies coordinate-invariance, which requires that $DLB(t;\hat{\mathbf{r}}\hat{\mathbf{p}}\hat{\mathbf{r}}^*,\hat{\mathbf{r}}\hat{\mathbf{q}}\hat{\mathbf{r}}^*) = \hat{\mathbf{r}}DLB(t;\hat{\mathbf{p}},\hat{\mathbf{q}})\hat{\mathbf{r}}^*$. We will see later that coordinate-invariance does not imply bi-invariance, thus bi-invariance is a stronger property. For proof of the left invariance it is sufficient to use distributivity of dual quaternions and the fact that $\|(1-t)\hat{\mathbf{r}}\hat{\mathbf{p}}+t\hat{\mathbf{r}}\hat{\mathbf{q}}\| = \|\hat{\mathbf{r}}\|\|(1-t)\hat{\mathbf{p}}+t\hat{\mathbf{q}}\| = \|(1-t)\hat{\mathbf{p}}+t\hat{\mathbf{q}}\|$ (recall that $\hat{\mathbf{r}}$ is a unit dual quaternion). $DLB(t;\hat{\mathbf{r}}\hat{\mathbf{p}},\hat{\mathbf{r}}\hat{\mathbf{q}})$ can thus be written as

$$\frac{(1-t)\hat{\mathbf{r}}\hat{\mathbf{p}}+t\hat{\mathbf{r}}\hat{\mathbf{q}}}{\|(1-t)\hat{\mathbf{r}}\hat{\mathbf{p}}+t\hat{\mathbf{r}}\hat{\mathbf{q}}\|} = \hat{\mathbf{r}}\frac{(1-t)\hat{\mathbf{p}}+t\hat{\mathbf{q}}}{\|(1-t)\hat{\mathbf{p}}+t\hat{\mathbf{q}}\|} = \hat{\mathbf{r}}DLB(t;\hat{\mathbf{p}},\hat{\mathbf{q}})$$

Proof of the right invariance is a direct analogy of the proof above. The left invariance of both DLB and ScLERP simplifies their comparison (left invariance of ScLERP is proven in Lemma 2 in the Appendix A). Instead of comparing $DLB(t;\hat{\mathbf{p}},\hat{\mathbf{q}})$ directly with $ScLERP(t;\hat{\mathbf{p}},\hat{\mathbf{q}})$, we rewrite them as $\hat{\mathbf{p}}DLB(t;1,\hat{\mathbf{p}}^*\hat{\mathbf{q}})$ and $\hat{\mathbf{p}}ScLERP(t;1,\hat{\mathbf{p}}^*\hat{\mathbf{q}})$, which is correct because of left invariance (1 is simply a real unit – when viewed as a dual quaternion, it corresponds to a rigid transformation with zero angle and zero translation). Since $\hat{\mathbf{p}}$ is the same in both expressions, it is sufficient to compare just $DLB(t;1,\hat{\mathbf{p}}^*\hat{\mathbf{q}})$ with $ScLERP(t;1,\hat{\mathbf{p}}^*\hat{\mathbf{q}})$, which is an easier problem. As $\hat{\mathbf{p}}^*\hat{\mathbf{q}}$ is a unit dual quaternion, it can be written as $\hat{\mathbf{p}}^*\hat{\mathbf{q}} = \cos\frac{\hat{\alpha}}{2} + \hat{\mathbf{n}}\sin\frac{\hat{\alpha}}{2}$. This enables us to derive

$$DLB(t;1,\hat{\mathbf{p}}^*\hat{\mathbf{q}}) = \frac{1-t+t\hat{\mathbf{p}}^*\hat{\mathbf{q}}}{\|1-t+t\hat{\mathbf{p}}^*\hat{\mathbf{q}}\|} = \frac{1-t+t\cos(\frac{\hat{\alpha}}{2}) + \hat{\mathbf{n}}t\sin(\frac{\hat{\alpha}}{2})}{\|1-t+t\hat{\mathbf{p}}^*\hat{\mathbf{q}}\|}$$

$$ScLERP(t;1,\hat{\mathbf{p}}^*\hat{\mathbf{q}}) = 1(1^*\hat{\mathbf{p}}^*\hat{\mathbf{q}})^t = (\hat{\mathbf{p}}^*\hat{\mathbf{q}})^t = \cos(t\frac{\hat{\alpha}}{2}) + \hat{\mathbf{n}}\sin(t\frac{\hat{\alpha}}{2})$$

from which we see that both DLB and ScLERP use the same, constant screw axis $\hat{\mathbf{n}}$. This means that DLB is a shortest path interpolation. Thus, the only difference between DLB and ScLERP is in the angle of rotation and amount of translation. We can compute an upper bound of this difference by computing the extremes of function $f(t) = \frac{1-t+t\cos(\frac{\hat{\alpha}}{2})}{\|1-t+t\hat{\mathbf{p}}^*\hat{\mathbf{q}}\|} - \cos(t\frac{\hat{\alpha}}{2})$ (the difference between the scalar parts of DLB and ScLERP). Function $f(t)$ is actually independent of $\hat{\mathbf{p}}^*\hat{\mathbf{q}}$, as can be shown by simplification of $\|1-t+t\hat{\mathbf{p}}^*\hat{\mathbf{q}}\|$. Computation of the maxima of $f(t)$ is not difficult but is a lengthy mathematical analysis. We present only results derived using Maple [Char et al. 1983]. The angles of rotation in DLB and ScLERP always differ by less than 8.15 degrees (note that this is in accordance with the results of [Kavan and Zara 2005]). The amount of translation always differs by less than 15% of the translation present in $\hat{\mathbf{p}}^*\hat{\mathbf{q}}$. Note that those results are *upper* bounds. In practice the difference is much smaller. To conclude, DLB is coordinate invariant, shortest path and "almost" constant speed. DLB works also for multiple rigid body transformations represented by unit dual quaternions $\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_n$ with convex weights $w_1,\ldots,w_n$. For brevity, we write the convex blending weights as a vector $\mathbf{w} = (w_1,\ldots,w_n)$.

$$DLB(\mathbf{w};\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_n) = \frac{w_1\hat{\mathbf{q}}_1 + \ldots + w_n\hat{\mathbf{q}}_n}{\|w_1\hat{\mathbf{q}}_1 + \ldots + w_n\hat{\mathbf{q}}_n\|}$$

(Formally, according to this convention, we should have written $DLB((1-t,t);\hat{\mathbf{p}},\hat{\mathbf{q}})$ instead of just $DLB(t;\hat{\mathbf{p}},\hat{\mathbf{q}})$ earlier. However, for conciseness we prefer the latter notation.) DLB is an approximate but fast solution to the problem of rigid transformation blending.

Our next goal is to derive an algorithm that converges to an exact solution. This problem has been solved for 3D rotations by spherical averages [Buss and Fillmore 2001]. It is not possible to simply apply spherical averages for unit dual quaternions, because the set of unit dual quaternions $\hat{Q}_1$ is not a hypersphere (instead, it is a set of tangent planes of a hypersphere [McCarthy 1990]). Fortunately, it has been shown that the Buss and Fillmore's idea can be generalized to other manifolds [Govindu 2004]. The latter paper is based on the theory of matrix groups and Lie algebras. We propose a similar algorithm based on dual quaternions, which is more efficient – due to the simple logarithm and exponential for dual quaternions. The result is an algorithm which we call Dual quaternion Iterative Blending (DIB):

Input: Unit dual quaternions $\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_n$, convex weights $\mathbf{w} = (w_1,\ldots,w_n)$, desired precision p
Output: Blended unit dual quaternion $\hat{\mathbf{b}}$
  $\hat{\mathbf{b}} = DLB(\mathbf{w};\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_n)$
  repeat
    $\hat{\mathbf{x}} = \sum_{i=1}^{n} w_i \log(\hat{\mathbf{b}}^*\hat{\mathbf{q}}_i)$

$\hat{\mathbf{b}} = \hat{\mathbf{b}}\exp(\hat{\mathbf{x}})$
until $\|\hat{\mathbf{x}}\| < p$
return $\hat{\mathbf{b}}$

A detailed mathematical discussion of this algorithm is beyond the scope of this paper (c.f. the complexity of discussion of a simpler algorithm [Buss and Fillmore 2001]). In practice, the DIB algorithm converges very quickly, typically in 1 to 4 steps. An intuitive explanation of this algorithm is shown in Figure 4.

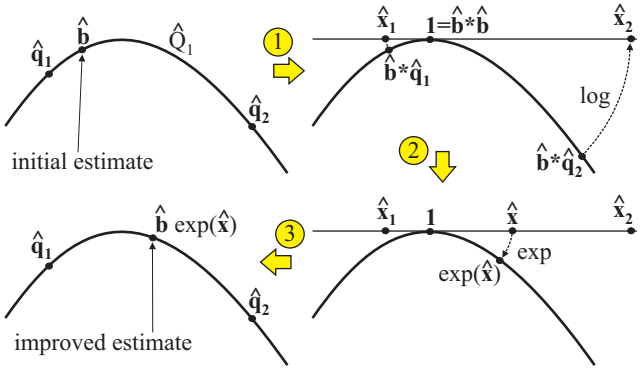


Figure 4: Illustration of one iteration of the DIB algorithm for $n = 2$ in a 2D slice of an 8-dimensional manifold $\hat{Q}_1$ (which is not a hypersphere). First, the input dual quaternions are left-multiplied by $\hat{\mathbf{b}}^*$, which maps the initial estimate $\hat{\mathbf{b}}$ onto the identity. The logarithm mapping then transforms $\hat{\mathbf{b}}^*\hat{\mathbf{q}}_1$, $\hat{\mathbf{b}}^*\hat{\mathbf{q}}_2$ into the tangent space of Q_1 at the identity, giving $\hat{\mathbf{x}}_1 = \log(\hat{\mathbf{b}}^*\hat{\mathbf{q}}_1)$, $\hat{\mathbf{x}}_2 = \log(\hat{\mathbf{b}}^*\hat{\mathbf{q}}_2)$. The blended value $\hat{\mathbf{x}} = w_1\hat{\mathbf{x}}_1 + w_2\hat{\mathbf{x}}_2$ is computed and projected back by the exponential mapping. Finally, multiplication $\hat{\mathbf{b}}\exp(\hat{\mathbf{x}})$ yields the unit dual quaternion closer to the exact solution.

The bi-invariance of DIB (and thus also the coordinate-invariance), follows from the bi-invariance of DLB and Lemma 1 in the Appendix A. The comparison of ScLERP with DIB is quite surprising. Not only is the result of DIB exactly equivalent to the result of ScLERP, but DIB finds this solution in just a single iteration (the second pass through the loop finds that $\|\hat{\mathbf{x}}\|$ is zero and performs no update of $\hat{\mathbf{b}}$). We prove this interesting property in Lemma 3 in the Appendix A. An immediate consequence is that DIB is also constant speed and shortest path. The DIB algorithm therefore presents an exact rigid transformation blending, but it can be slower to compute than the closed-form DLB.

5 Comparison with Other Methods

Besides the DLB and DIB algorithms described in the previous section, several alternative methods for blending of rigid transformations have been described. This section compares the properties of these blending algorithms in terms of coordinate-invariance (bi-invariance), constant speed, shortest path, and computation time in terms of FLOPS. Those properties for dual quaternion based algorithms were discussed in Section 4. Properties of previous rigid transformation blending algorithms are discussed below and summarized in Table 1.

Linear combination of matrices (Lin). Because of the distributivity of matrix products, this method is naturally bi-invariant (and thus also coordinate invariant). Constant speed and shortest path properties do not hold, because their definition assumes that the blended transformation will be rigid – which is not the case in linear combination of matrices.

Decomposition to rotation and translation (Dec). As discussed already in Section 2, this solution is coordinate dependent. The remaining properties follow directly from properties of linear blending of translations and spherical averages of rotations [Buss and Fillmore 2001].

Blending of matrix logarithms (Log). Log-matrix blending followed by exponentiation [Alexa 2002] is coordinate invariant, because an analogy of Lemma 1 is true also for matrices: $\exp(TMT^{-1}) = T\exp(M)T^{-1}$, $\log(TMT^{-1}) = T\log(M)T^{-1}$ [Moakher 2002]. However, it is not bi-invariant, because $\exp(TM) \neq T\exp(M)$. Even though this method has a nice geometric interpretation, it is neither shortest path [Bloom et al. 2004] nor constant speed [Alexa 2002] (note that Bloom et al. [2004] incorrectly state that it is constant speed). We prove that the speed of log-matrix blending is really not constant in Appendix B.

The DIB algorithm from the previous section suggests how Alexa's blending can be upgraded to be both constant speed and shortest path: if we replace dual quaternions by matrices, and the initial value of $\hat{\mathbf{b}}$ by the identity (instead of DLB), we obtain an iterative version of Alexa's method. In the first iteration, the modified DIB algorithm computes the same result as Alexa's original algorithm. In subsequent iterations, the modified DIB algorithm converges to a constant speed and shortest path solution. This way, we obtain an arbitrary-precision solution to the blending problem for a more general class of transformations. However, this comes at a cost: the matrix exponential and logarithm routines in this case require an iterative numerical solution. As several iterations are required, the numerical errors of log and exp routines accumulate. Therefore, if we deal only with rigid transformations, it is both more robust and efficient to apply the original DIB algorithm, which uses simple and efficient dual quaternion exp and log (see end of Section 3).

	Lin.	Dec.	Log.	ScLERP	DLB	DIB
Rigidity	-	+	+	+	+	+
Bi-inv.	+	-	-	+	+	+
Coord-inv.	+	-	+	+	+	+
$n > 2$	+	+	+	-	+	+
Const-spd.	-	+	-	+	-	+
Shortest	-	+	-	+	+	+
FLOPS	$23n$	N/A	$160+$ $104n$	240	$65+$ $49n$	N/A

Table 1: Properties of different rigid transformation blending algorithms (top to bottom): preserving of rigid transformations, bi-invariance, coordinate-invariance, support of more than two rigid transformations, constant speed, shortest path, and number of FLOPS for blending n rigid transformations. FLOPS for Dec. and DIB depend on the actual input and on desired precision, because the algorithms are iterative.

We conclude that, for two rigid transformations, the optimal choice is ScLERP. If we need a precise solution for more than two transformations, we have to employ the iterative DIB. The FLOPS of log-matrix blending reported in Table 1 refer to an optimization, which is restricted to rigid transformations and employs a closed-form Rodrigues formula for rigid transformations [Murray et al. 1994; Angelidis 2004]. However, even though it is closed-form, the Rodrigues formula is still more difficult to compute than the simple dual quaternion exp and log. We see that there is no reason to apply log-matrix blending for rigid transformations, because DLB has better properties and faster runtime. In the calculation of FLOPS, we assume that both input and output are 4×4 homogeneous ma-

trices (specifically, the FLOPS for matrix↔dual quaternion conversions are included in Table 1; if an application works internally with quaternions instead of matrices, then the performance of our proposed algorithms is even better).

6 Case Study: Skinning

Blending of rigid transformations has many applications in computer graphics, as shown already by Alexa [2002]. One particular application, geometric skinning, has received a lot of research attention recently, so we have chosen this problem to demonstrate the behavior of dual quaternion algorithms in practice. In geometric skinning, a rigged virtual model (such as a character, animal or piece of cloth) is composed of a triangular mesh, a list L of rigid transformations, and weights for every vertex in the mesh. Traditionally, the list L is interpreted as a list of joint displacements from the reference to the animated skeleton. However, geometric skinning can be applied also for general deformable models without any skeleton, as has been shown recently [James and Twigg 2005]. The animation of the whole model is driven by the animation of individual transformations in L. The task of the skinning algorithm is to plausibly deform the skin for a given set of transformations. An arbitrary vertex $\mathbf{v}$ in the mesh is displaced as follows: let us suppose there are m transformations, $C_1,\ldots,C_m$ from L, influencing vertex $\mathbf{v}$, and the convex weights of vertex $\mathbf{v}$ are $w_1,\ldots,w_m$. A geometric skinning algorithm then computes B, the blended rigid transformation of $C_1,\ldots,C_m$ with weights $w_1,\ldots,w_m$. The final vertex position in the deformed skin is computed simply as $B\mathbf{v}$. This deformation method is very popular for its simplicity and speed. Note however, that if additional information is available, such as measurements of real subjects, better methods exist [Allen et al. 2002; Sand et al. 2003].

A number of approaches to geometric skinning have been recently presented, each one employing a different method of rigid transformation blending. Mohr and Gleicher [2003] apply Linear Blend Skinning (LBS), which uses an efficient linear combination of matrices, but exhibits anomalies such as the *candy wrapper* artifact (see Figure 6(a)). Better visual results can be achieved, but at the cost of using auxiliary joints and example skins. Hejl [2004] ignores blending of translation and performs linear quaternion blending for rotations. This works correctly only if the influencing transformations $C_1,\ldots,C_m$ of every vertex have a common fixed point, which is true only in simplified skeletal models, but not in general. Magnenat-Thalmann et al. [2004] apply Alexa's [2002] log-matrix blending to character skinning and cloth simulation. The problems of Alexa's algorithm discussed above manifest in non-natural skin deformations for certain skeletal postures (see Figure 1(a) and Figure 6(c)). Kavan and Zara's Spherical Blend Skinning (SBS) [2005] blends $(quaternion, translation)$ pairs and copes with the dependence on the coordinate system by computing the optimal origin of the coordinate system (called *rotation center*). Unfortunately, the optimization involves a complex and slow SVD (Singular Value Decomposition) algorithm. To make real-time execution possible, SVD is computed only for clusters of vertices. However, this introduces discontinuities in the deformed skin between individual clusters (see Figure 1(c)). As we see, no single previous methods is clearly superior – each presents trade-offs that must be considered.

We have applied our DLB algorithm to the problem of geometric skinning and have developed both CPU and GPU implementations. For our experiments, we used a skeletal model of a woman with 5002 vertices, 9253 triangles, and 54 joints. The average runtime performance is reported in Figure 5. We compared the speed with log-matrix blending (Log) and spherical blend skinning (SBS) only on a CPU, because no GPU implementation was available for these methods. The piece of cloth shown in Figure 1 has 6000 vertices, 12000 triangles and 49 joints (observe the discontinuity of spherical blending in Figure 1(c)). The measurements, together with the visual results in Figures 1 and 6, clearly show that dual quaternion blending is not only more accurate, but also more than twice as fast as both log-matrix and spherical blending. Dual quaternion skinning is slightly slower than linear blend skinning, but has the advantage of artifact elimination (Figure 6(a-b)) and usage of fewer registers – dual quaternions only need 8 floats instead of the 12 required by matrices, which is of particular benefit for the GPU implementation.

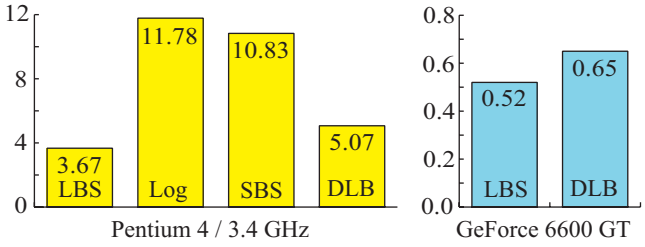


Figure 5: Average CPU/GPU runtimes for skin deformation of a woman model in milliseconds.

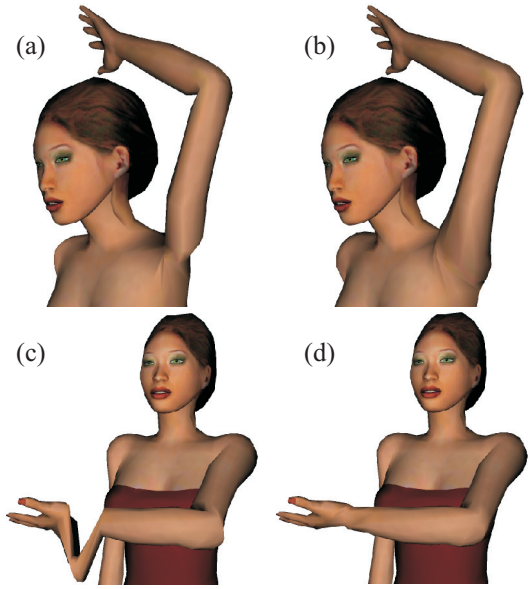


Figure 6: (a) Typical "candy wrapper" of linear blending, (b) solved by dual quaternion blending. (c) Non-shortest path of log-matrix blending implies non-natural deformations, (d) shortest-path dual quaternion blending.

7 Conclusions and Future Work

Rigid transformation blending algorithms based on dual quaternions exhibit more advantageous properties, and faster execution times than previous methods. We verified this both theoretically and on a practical case study of skinning. We believe that dual quaternion skinning is the algorithm that will finally replace linear blend skinning in computer games. In the future, the dual quaternion blending algorithms presented in this paper could be applied for example in:

- motion blending – extension of [Park et al. 2002]
- motion analysis & compression – as suggested by [Alexa 2002]
- spatial keyframing – extension of [Igarashi et al. 2005]
- computer vision – averaging of measured position and orientation of a rigid object
- graphics hardware – dual quaternions take 8 scalars to represent a rigid transformation instead of the 12 required for a matrix.

To conclude, one of the biggest advantages of dual quaternions is that they are based on classical quaternions, which are well-known in the computer graphics community. We believe that dual quaternions will soon become part of every 3D graphics library.

References

ALEXA, M. 2002. Linear combination of transformations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 380–387.

ALLEN, B., CURLESS, B., AND POPOVIC, Z. 2002. Articulated body deformation from range scan data. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 612–619.

ANGELIDIS, A., 2004. Hexanions: 6D space for twists. Technical report OUCS-2004-20, University of Otago.

BLOOM, C., BLOW, J., AND MURATORI, C., 2004. Errors and omissions in Marc Alexa's Linear combination of transformations. `http://www.cbloom.com/3d/techdocs/lcot_errors.pdf`.

BOTTEMA, O., AND ROTH, B. 1979. *Theoretical kinematics*. North-Holland Publishing Company, Amsterdam, New York, Oxford.

BREGLER, C., AND MALIK, J. 1998. Tracking people with twists and exponential maps. In *CVPR '98: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, Washington, DC, USA, 8.

BUSS, S. R., AND FILLMORE, J. P. 2001. Spherical averages and applications to spherical splines and interpolation. *ACM Trans. Graph. 20*, 2, 95–126.

CHAR, B., GEDDES, K., AND GONNET, G. 1983. The Maple symbolic computation system. *j-SIGSAM 17*, 3–4 (Aug./Nov.), 31–42.

CLIFFORD, W. 1882. *Mathematical Papers*. London, Macmillan.

DAM, E., KOCH, M., AND LILLHOLM, M., 1998. Quaternions, interpolation and animation. Technical Report DIKU-TR-98/5, University of Copenhagen.

DANIILIDIS, K. 1999. Hand-eye calibration using dual quaternions. *International Journal of Robotics Research 18*, 286–298.

EBERLY, D. 2001. *3D game engine design: a practical approach to real-time computer graphics*. Morgan Kaufmann Publishers Inc.

GE, Q. J., AND RAVANI, B. 1994. Computer aided geometric design of motion interpolants. *ASME Journal of Mechanical Design 116*, 3, 756–762.

GE, Q. J., VARSHNEY, A., MENON, J., AND CHANG, C. 1998. Double quaternion for motion interpolation. In *Proc. 1998 ASME Design Manufacturing Conference*.

GOVINDU, V. M. 2004. Lie-algebraic averaging for globally consistent motion estimation. In *CVPR (1)*, 684–691.

HEJL, J., 2004. Hardware skinning with quaternions. *Game Programming Gems 4*, Charles River Media, 487–495.

HILDENBRAND, D., FONTIJNE, D., PERWASS, C., AND DORST, L., 2004. Geometric algebra and its application to computer graphics. EG 2004 tutorial #3.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. Spatial keyframing for performance-driven animation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 107–115.

JAMES, D. L., AND TWIGG, C. D. 2005. Skinning mesh animations. *ACM Trans. Graph. 24*, 3, 399–407.

JOHNSON, M. P. 2003. *Exploiting Quaternions to Support Expressive Interactive Character Motion*. PhD thesis, MIT.

JUTTLER, B. 1994. Visualization of moving objects using dual quaternion curves. *Computers & Graphics 18*, 3, 315–326.

KAVAN, L., AND ZARA, J. 2005. Spherical blend skinning: A real-time deformation of articulated models. In *2005 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press, 9–16.

LEE, J., AND SHIN, S. Y. 2002. General construction of time-domain filters for orientation data. *IEEE Transactions on Visualization and Computer Graphics 8*, 2, 119–128.

LUCIANO, C., AND BANERJEE, P. 2000. Avatar kinematics modeling for telecollaborative virtual environments. In *WSC '00: Proceedings of the 32nd conference on Winter simulation*, Society for Computer Simulation International, San Diego, CA, USA, 1533–1538.

MAGNENAT-THALMANN, N., CORDIER, F., SEO, H., AND PAPAGIANAKIS, G. 2004. Modeling of bodies and clothes for virtual environments. In *CW '04: Proceedings of the 2004 International Conference on Cyberworlds (CW'04)*, IEEE Computer Society, 201–208.

MCCARTHY, J. M. 1990. *Introduction to theoretical kinematics*. MIT Press, Cambridge, MA, USA.

MOAKHER, M. 2002. Means and averaging in the group of rotations. *SIAM Journal on Matrix Analysis and Applications 24*, 1, 1–16.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3, 562–568.

MURRAY, R. M., SASTRY, S. S., AND ZEXIANG, L. 1994. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 413–414.

PARK, S. I., SHIN, H. J., AND SHIN, S. Y. 2002. On-line locomotion generation based on motion blending. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, 105–111.

PEREZ, A., AND MCCARTHY, J. M. 2004. Dual quaternion synthesis of constrained robotic systems. *Journal of Mechanical Design 126*, 425–435.

SAND, P., MCMILLAN, L., AND POPOVIC, J. 2003. Continuous capture of skin deformation. *ACM Trans. Graph. 22*, 3, 578–586.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, ACM Press, 245–254.

Appendix A

Lemma 1. *Let* $\hat{\mathbf{q}} = \cos\frac{\hat{\theta}}{2} + \hat{\mathbf{s}}\sin\frac{\hat{\theta}}{2}$*, where* $\hat{\mathbf{q}}, \hat{\mathbf{s}}$ *are unit dual quaternions, and* $\hat{\mathbf{s}}$ *has zero scalar part. Then for any unit dual quaternion* $\hat{\mathbf{m}}$*, both of the following equations are true:*

$$1)\ \exp(\hat{\mathbf{m}}\hat{\mathbf{s}}\frac{\hat{\theta}}{2}\hat{\mathbf{m}}^*) = \hat{\mathbf{m}}\exp(\hat{\mathbf{s}}\frac{\hat{\theta}}{2})\hat{\mathbf{m}}^*,\ 2)\ \log(\hat{\mathbf{m}}\hat{\mathbf{q}}\hat{\mathbf{m}}^*) = \hat{\mathbf{m}}\log(\hat{\mathbf{q}})\hat{\mathbf{m}}^*$$

Proof. Since the scalar part of $\hat{\mathbf{s}}$ is zero, the same is true for the scalar part of $\hat{\mathbf{m}}\hat{\mathbf{s}}\frac{\hat{\theta}}{2}\hat{\mathbf{m}}^*$, as can be shown by direct computation (we employed Maple [Char et al. 1983]). This means that the exp on the left hand side of 1) is well defined and, according to its definition,

$$\exp(\hat{\mathbf{m}}\hat{\mathbf{s}}\frac{\hat{\theta}}{2}\hat{\mathbf{m}}^*) = \cos\frac{\hat{\theta}}{2} + \hat{\mathbf{m}}\hat{\mathbf{s}}\sin\frac{\hat{\theta}}{2}\hat{\mathbf{m}}^* = \hat{\mathbf{m}}(\cos\frac{\hat{\theta}}{2} + \hat{\mathbf{s}}\sin\frac{\hat{\theta}}{2})\hat{\mathbf{m}}^*$$

because a dual number always commutes with a dual quaternion and $\hat{\mathbf{m}}\hat{\mathbf{m}}^* = 1$. This shows the first equation. The proves of the second one is similar:

$$\hat{\mathbf{m}}\hat{\mathbf{q}}\hat{\mathbf{m}}^* = \hat{\mathbf{m}}(\cos\frac{\hat{\theta}}{2} + \hat{\mathbf{s}}\sin\frac{\hat{\theta}}{2})\hat{\mathbf{m}}^* = \cos\frac{\hat{\theta}}{2} + \hat{\mathbf{m}}\hat{\mathbf{s}}\hat{\mathbf{m}}^*\sin\frac{\hat{\theta}}{2}$$

Therefore $\log(\hat{\mathbf{m}}\hat{\mathbf{q}}\hat{\mathbf{m}}^*) = \hat{\mathbf{m}}\hat{\mathbf{s}}\frac{\hat{\theta}}{2}\hat{\mathbf{m}}^* = \hat{\mathbf{m}}\log(\hat{\mathbf{q}})\hat{\mathbf{m}}^*$, as we wanted to prove. □

Lemma 2. *ScLERP is bi-invariant, that is for any unit dual quaternions* $\hat{\mathbf{r}}, \hat{\mathbf{p}}, \hat{\mathbf{q}}$ *and any interpolation parameter* $t \in [0,1]$*, both of the following equations are true:*

$$ScLERP(t; \hat{\mathbf{r}}\hat{\mathbf{p}}, \hat{\mathbf{r}}\hat{\mathbf{q}}) = \hat{\mathbf{r}}ScLERP(t; \hat{\mathbf{p}}, \hat{\mathbf{q}})$$
$$ScLERP(t; \hat{\mathbf{p}}\hat{\mathbf{r}}, \hat{\mathbf{q}}\hat{\mathbf{r}}) = ScLERP(t; \hat{\mathbf{p}}, \hat{\mathbf{q}})\hat{\mathbf{r}}$$

Proof. The left invariance is easy, because $ScLERP(t; \hat{\mathbf{r}}\hat{\mathbf{p}}, \hat{\mathbf{r}}\hat{\mathbf{q}}) = \hat{\mathbf{r}}\hat{\mathbf{p}}(\hat{\mathbf{p}}^*\hat{\mathbf{r}}^*\hat{\mathbf{r}}\hat{\mathbf{q}})^t = \hat{\mathbf{r}}\hat{\mathbf{p}}(\hat{\mathbf{p}}^*\hat{\mathbf{q}})^t = \hat{\mathbf{r}}ScLERP(t; \hat{\mathbf{p}}, \hat{\mathbf{q}})$. Proving the right invariance is a little more tricky: $ScLERP(t; \hat{\mathbf{p}}\hat{\mathbf{r}}, \hat{\mathbf{q}}\hat{\mathbf{r}}) = \hat{\mathbf{p}}\hat{\mathbf{r}}(\hat{\mathbf{r}}^*\hat{\mathbf{p}}^*\hat{\mathbf{q}}\hat{\mathbf{r}})^t$. It is now sufficient to show that $(\hat{\mathbf{r}}^*\hat{\mathbf{p}}^*\hat{\mathbf{q}}\hat{\mathbf{r}})^t = \hat{\mathbf{r}}^*(\hat{\mathbf{p}}^*\hat{\mathbf{q}})^t\hat{\mathbf{r}}$, because this gives us $\hat{\mathbf{p}}\hat{\mathbf{r}}(\hat{\mathbf{r}}^*\hat{\mathbf{p}}^*\hat{\mathbf{q}}\hat{\mathbf{r}})^t = \hat{\mathbf{p}}\hat{\mathbf{r}}\hat{\mathbf{r}}^*(\hat{\mathbf{p}}^*\hat{\mathbf{q}})^t\hat{\mathbf{r}} = ScLERP(t; \hat{\mathbf{p}}, \hat{\mathbf{q}})\hat{\mathbf{r}}$. However, the power can be written as $(\hat{\mathbf{r}}^*\hat{\mathbf{p}}^*\hat{\mathbf{q}}\hat{\mathbf{r}})^t = \exp(t\log(\hat{\mathbf{r}}^*\hat{\mathbf{p}}^*\hat{\mathbf{q}}\hat{\mathbf{r}}))$. Thanks to Lemma 1, we can derive that $\exp(t\log(\hat{\mathbf{r}}^*\hat{\mathbf{p}}^*\hat{\mathbf{q}}\hat{\mathbf{r}})) = \exp(t\hat{\mathbf{r}}^*\log(\hat{\mathbf{p}}^*\hat{\mathbf{q}})\hat{\mathbf{r}}) = \hat{\mathbf{r}}^*\exp(t\log(\hat{\mathbf{p}}^*\hat{\mathbf{q}}))\hat{\mathbf{r}} = \hat{\mathbf{r}}^*(\hat{\mathbf{p}}^*\hat{\mathbf{q}})^t\hat{\mathbf{r}}$. □

Lemma 3. *For the case of two rotations, the* $DIB(t; \hat{\mathbf{p}}, \hat{\mathbf{q}})$ *algorithm converges in a single iteration with result* $ScLERP(t; \hat{\mathbf{p}}, \hat{\mathbf{q}})$.

Proof. Thanks to the bi-invariance of DIB, we can perform the same simplification as when comparing DLB with ScLERP, that is instead of $DIB(t; \hat{\mathbf{p}}, \hat{\mathbf{q}})$ and $ScLERP(t; \hat{\mathbf{p}}, \hat{\mathbf{q}})$, compare $DIB(t; 1, \hat{\mathbf{p}}^*\hat{\mathbf{q}})$ and $ScLERP(t; 1, \hat{\mathbf{p}}^*\hat{\mathbf{q}})$. Let us denote the initial value of $\hat{\mathbf{b}}$ as $\hat{\mathbf{b}}_0 = DLB(t; 1, \hat{\mathbf{p}}^*\hat{\mathbf{q}})$. With weight vector $\mathbf{w} = (1-t, t)$ as the input, the DIB computes in the first iteration $\hat{\mathbf{b}}_1 = \hat{\mathbf{b}}_0\exp((1-t)\log(\hat{\mathbf{b}}_0^*) + t\log(\hat{\mathbf{b}}_0^*\hat{\mathbf{p}}^*\hat{\mathbf{q}}))$. As shown in Section 4, the only difference between *DLB* and *ScLERP* is in the angle of rotation and amount of translation. It means that $\hat{\mathbf{b}}_0 = (\hat{\mathbf{p}}^*\hat{\mathbf{q}})^{\hat{u}}$ for some dual number $\hat{u}$. Since for unit dual quaternions conjugation is the same as inverse, we can write $\hat{\mathbf{b}}_1 = (\hat{\mathbf{p}}^*\hat{\mathbf{q}})^{\hat{u}}\exp((1-t)\log((\hat{\mathbf{p}}^*\hat{\mathbf{q}})^{-\hat{u}}) + t\log((\hat{\mathbf{p}}^*\hat{\mathbf{q}})^{1-\hat{u}})) = (\hat{\mathbf{p}}^*\hat{\mathbf{q}})^{\hat{u}}\exp((t-\hat{u})\log(\hat{\mathbf{p}}^*\hat{\mathbf{q}})) = (\hat{\mathbf{p}}^*\hat{\mathbf{q}})^{\hat{u}+t-\hat{u}} = (\hat{\mathbf{p}}^*\hat{\mathbf{q}})^t = ScLERP(t; 1, \hat{\mathbf{p}}^*\hat{\mathbf{q}})$. In the following iteration, the DIB algorithm computes $\hat{\mathbf{x}} = (1-t)\log\hat{\mathbf{b}}_1^* + t\log(\hat{\mathbf{b}}_1^*\hat{\mathbf{p}}^*\hat{\mathbf{q}}) = (1-t)\log((\hat{\mathbf{p}}^*\hat{\mathbf{q}})^{-t}) + t\log((\hat{\mathbf{p}}^*\hat{\mathbf{q}})^{1-t}) = (t^2 - t + t - t^2)\log(\hat{\mathbf{p}}^*\hat{\mathbf{q}}) = 0$ and thus the algorithm $DIB(t; 1, \hat{\mathbf{p}}^*\hat{\mathbf{q}})$ terminates with zero error and returns $\hat{\mathbf{b}}_1\exp(0) = \hat{\mathbf{b}}_1 = ScLERP(t; 1, \hat{\mathbf{p}}^*\hat{\mathbf{q}})$. Multiplication from left by $\hat{\mathbf{p}}$ and using the left-invariance yields $DIB(t; \hat{\mathbf{p}}, \hat{\mathbf{q}}) = ScLERP(t; \hat{\mathbf{p}}, \hat{\mathbf{q}})$. □

Appendix B

In this appendix, we prove the non-constant speed of log-matrix blending [Alexa 2002]. This problem has an interesting history. The author of log-matrix blending claims, without proof, that his method is not constant speed [Alexa 2002]. Subsequently, a critique of log-matrix blending is posted on-line [Bloom et al. 2004], which points out mistakes in Alexa's paper [Alexa 2002]. Among many good insights, it unfortunately mentions also the fact that log-matrix blending actually *is* constant speed. This is not true, as we prove below.

Background on Log-matrix Blending

Before we start with the actual proof, we review log-matrix blending with a special focus on rotation blending. This gives us a deeper insight into the problem.

Let us consider a simple situation of two 3×3 rotation matrices R_0 and R_1. Let R_t be a blending (interpolation) between those two matrices, i.e., a matrix which for $t = 0$ becomes R_0, for $t = 1$ becomes R_1 and for $t \in (0,1)$ is a valid rotation matrix. What we informally referred as speed in the above, is actually an angular velocity of R_t. The formula expressing angular velocity in the body (moving) coordinate system is

$$M(\omega_t) = R_t^{-1}\frac{\partial R_t}{\partial t} \tag{3}$$

see [Murray et al. 1994]. Alternatively, we could also use a similar formula for angular velocity expressed in the spatial coordinate system. This angular velocity differs only by the reference coordinate system, and thus its magnitude (which we aim to compute) is the same. In our analysis, we will work with the body angular velocity, although we could equally well work with the spatial angular velocity. In Formula (3), $M(\mathbf{a})$ is a function mapping vector $\mathbf{a} = (a_1, a_2, a_3)$ to an anti-symmetric matrix

$$M(\mathbf{a}) = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix} \tag{4}$$

The multiplication of any vector $\mathbf{x} = (x_1, x_2, x_3)$ by this matrix corresponds to the cross product, i.e., $M(\mathbf{a})\mathbf{x} = \mathbf{a} \times \mathbf{x}$. Therefore, the vector ω_t in Formula (3) is the common vector representation of angular velocity. The anti-symmetric matrix $M(\mathbf{a})$ is connected with matrix logarithms. If R is a rotation about axis $\mathbf{a}/\|\mathbf{a}\|$ with angle $\|\mathbf{a}\|$, then its logarithm $\log R = M(\mathbf{a})$. This gives us an intuitive explanation of the rotation matrix logarithm. To verify this fact, consider a differential equation describing rotation of a point $\mathbf{p}$ at time t with angular velocity $\mathbf{a}$:

$$\frac{\partial \mathbf{p}(t)}{\partial t} = \mathbf{a} \times \mathbf{p}(t) = M(\mathbf{a})\mathbf{p}(t)$$

The solution of this differential equation can be expressed using the matrix exponential

$$\mathbf{p}(t) = \exp(M(\mathbf{a})t)\mathbf{p}(0) \tag{5}$$

where $\mathbf{p}(0)$ is the initial condition, i.e., the position of the point at time 0. We can observe that the term $\exp(M(\mathbf{a})t)$ in Formula (5) is nothing but the matrix of rotation about axis $\mathbf{a}/\|\mathbf{a}\|$ with angle $t\|\mathbf{a}\|$. Therefore, the matrix R describing rotation about axis $\mathbf{a}/\|\mathbf{a}\|$ with angle $\|\mathbf{a}\|$ can be written as $R = \exp(M(\mathbf{a}))$. From this, it immediately follows that $\log R = M(\mathbf{a})$, as we wanted to show.

In the following, R_t will denote the result of log-matrix blending, given according to [Alexa 2002] as

$$R_t = \exp((1-t)\log R_0 + t\log R_1) \tag{6}$$

where exp and log denote the matrix exponential and logarithm. The geometrical interpretation of matrix logarithm gives us an insight into what the log-matrix blending (limited to rotations) actually does: linear blending of the axis-angle representation of rotations.

Log-Matrix Blending in Maple

To show that log-matrix blending is not constant speed, it is sufficient to find two rotation matrices R_0, R_1, and show that the magnitude of angular velocity of their blend R_t, i.e., $\|\omega_t\|$ according to Formula (3), is not a constant function. Let us define R_0 as a rotation about axis $(1,0,0)$ with angle 1 radian, and matrix R_1 as a rotation about axis $(1/\sqrt{2}, 1/\sqrt{2}, 0)$ with angle $\sqrt{2}$ radians. This choice simplifies the following computations. The logarithms of R_0 and R_1 are

$$\log R_0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, \quad \log R_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$$

and therefore

$$(1-t)\log R_0 + t\log R_1 = \begin{pmatrix} 0 & 0 & t \\ 0 & 0 & -1 \\ -t & 1 & 0 \end{pmatrix}$$

We denote this matrix as $L_t := (1-t)\log R_0 + t\log R_1$. The next step is to compute the exponential of matrix L_t. In order to avoid numerical inaccuracies, we proceed with symbolic computations in Maple [Char et al. 1983] (it would also be possible to use the Rodrigues formula [Murray et al. 1994], but the equations quickly become awkward for manual derivations). Computing the exponential of L_t yields

$$\exp L_t = \begin{pmatrix} \frac{1+t^2\cos(\sqrt{1+t^2})}{1+t^2} & -\frac{t\left(\cos(\sqrt{1+t^2})-1\right)}{1+t^2} & \frac{t\sin(\sqrt{1+t^2})}{\sqrt{1+t^2}} \\ -\frac{t\left(\cos(\sqrt{1+t^2})-1\right)}{1+t^2} & \frac{t^2+\cos(\sqrt{1+t^2})}{1+t^2} & -\frac{\sin(\sqrt{1+t^2})}{\sqrt{1+t^2}} \\ -\frac{t\sin(\sqrt{1+t^2})}{\sqrt{1+t^2}} & \frac{\sin(\sqrt{1+t^2})}{\sqrt{1+t^2}} & \cos(\sqrt{1+t^2}) \end{pmatrix}$$

which is the result of log-matrix blending, denoted as $R_t := \exp L_t$. Now we compute the inverse and derivative of R_t, whose multiplication gives the angular velocity matrix $M(\omega_t)$, according to Formula (3). The resulting matrix is indeed anti-symmetric, as expected, and therefore has the structure from Formula (4). If we extract the angular velocity vector according to Formula (4), we obtain vector $\omega_t = (\omega_{t,0}, \omega_{t,1}, \omega_{t,2})$, where

$$\begin{aligned} \omega_{t,0} &= -\frac{\left(-t^2 - 1 + \sqrt{1+t^2}\sin(\sqrt{1+t^2})\right)t}{\left(1+t^2\right)^2} \\ \omega_{t,1} &= \frac{t^4 + t^2 + \sqrt{1+t^2}\sin(\sqrt{1+t^2})}{\left(1+t^2\right)^2} \\ \omega_{t,2} &= \frac{\cos(\sqrt{1+t^2}) - 1}{1+t^2} \end{aligned}$$

After computing the magnitude of vector ω_t we get

$$\|\omega_t\| = \sqrt{\frac{t^4 + t^2 - 2\cos(\sqrt{1+t^2}) + 2}{\left(1+t^2\right)^2}}$$

Obviously, this function is not constant for $t \in [0,1]$, see also its graph in Figure 7.

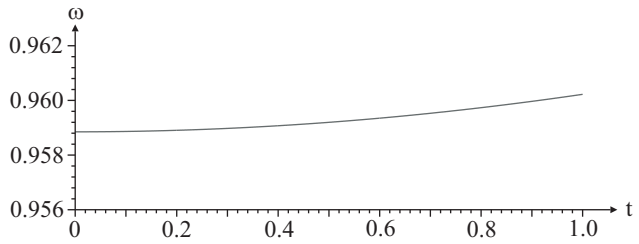


Figure 7: Graph of $\|\omega_t\|$ for $t \in [0,1]$