Geometric Skinning with Approximate Dual Quaternion Blending

Ladislav Kavan*[1] Steven Collins[1] Jiří Žára[2] Carol O'Sullivan[1]

[1]Trinity College Dublin, [2]Czech Technical University in Prague

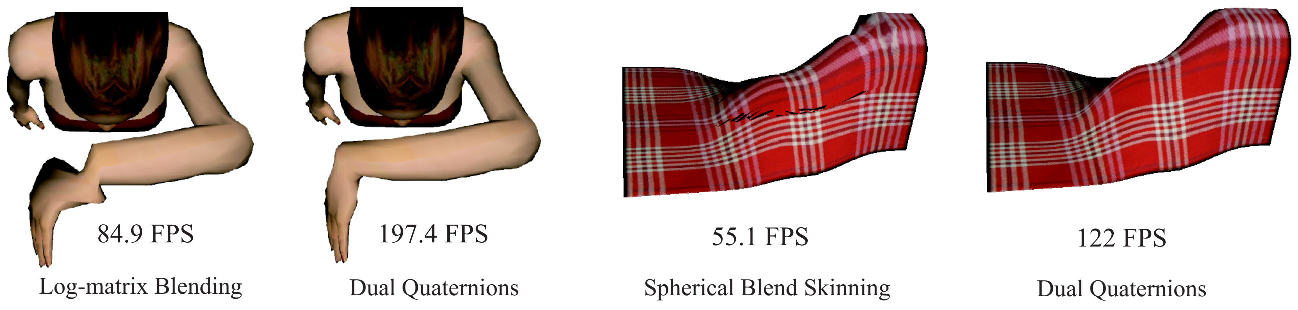


Figure 1: A comparison of dual quaternion skinning with previous methods: log-matrix blending [Cordier and Magnenat-Thalmann 2005] and spherical blend skinning [Kavan and Žára 2005]. The proposed approach not only eliminates artifacts, but is also much easier to implement and more than twice as fast.

Abstract

Skinning of skeletally deformable models is extensively used for real-time animation of characters, creatures and similar objects. The standard solution, linear blend skinning, has some serious drawbacks that require artist intervention. Therefore, a number of alternatives have been proposed in recent years. All of them successfully combat some of the artifacts, but none challenge the simplicity and efficiency of linear blend skinning. As a result, linear blend skinning is still the number one choice for the majority of developers. In this paper, we present a novel skinning algorithm based on linear combination of dual quaternions. Even though our proposed method is approximate, it does not exhibit any of the artifacts inherent in previous methods and still permits an efficient GPU implementation. Upgrading an existing animation system from linear to dual quaternion skinning is very easy and has a relatively minor impact on run-time performance.

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: skinning, rigid transformations, blending, dual quaternions, linear combinations

1 Introduction

Skinning and skeletal animation is the technology behind character animation in many applications. In some situations, physically accurate skin deformation, which supports muscle bulging and dynamic effects, is desirable. In other situations, however, a fast algorithm capable of skinning multiple models interactively is needed, for example in videogames and crowd simulations.

The standard algorithm for low-cost skinning is known by many names: linear blend skinning, vertex blending, skeletal subspace deformation or enveloping. It is sometimes used not only for skin deformation (as the name suggests) but also to animate other deforming elements, for example cloth, because it is considerably faster than physically based cloth simulation [Cordier and Magnenat-Thalmann 2005]. The basic principle is that skinning transformations are represented by matrices and blended linearly. It is very well known that the direct linear combination of matrices is a troublesome way of blending transformations. This produces artifacts in the deformed skin, even if we restrict the skinning transformations to rigid ones (i.e., composition of a rotation and translation). In spite of these shortcomings, linear blending is still a very popular skinning method, but perhaps only because there is no simple alternative.

Recent previous work suggests converting rigid transformation matrices to $(quaternion, translation)$ pairs and blending them instead of their matrix equivalents [Hejl 2004; Kavan and Žára 2005]. This works, but at a cost: Hejl's algorithm [2004] imposes constraints on the model's rigging (specifically, a vertex can only be influenced by neighbouring bones, otherwise artifacts can occur). This could be inconvenient, because linear blending has no such constraints (and it is exploited for many 3D models). Kavan and Žára's method [2005] does not have this restriction, but uses a complex and computationally expensive Singular Value Decomposition scheme. Obviously, the simplicity of linear blend skinning is lost in both cases.

The representation of rigid transformations by matrices or $(quaternion, translation)$ pairs illustrates just two possible parameterizations of $SE(3)$, i.e., the group of rigid transformations. Nothing prevents us from blending, for example, 3-tuples $(axis, angle, translation)$ or pairs $(axis, translation \cdot \sin(angle))$. Even if we restrict ourselves to blending via linear combinations (motivated by efficiency and simplicity of implementation), we can construct infinitely many different blending methods just by considering different parameterizations of rigid transformations.

*e-mail: kavanl@cs.tcd.ie

A theoretically optimal rigid transformation blending method has been proposed previously [Govindu 2004]. The algorithm possesses all desired mathematical properties that guarantee correct skinning, but unfortunately it is iterative and thus prohibitively slow for most real-time applications. Therefore, we propose instead a closed-form approximation, based on dual quaternions – a generalization of regular quaternions first proposed in the nineteenth century [Clifford 1882].

These concepts can be elegantly illustrated in 2D Euclidean space. Assume we have 3 points $\mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{p}_3$ lying on a spherical arc (representing rotation about the origin), such as in Figure 2. Direct averaging of point coordinates (left) produces $\mathbf{p}_{avg}$ that no longer lies on the arc. This is the reason for artifacts in linear blend skinning – in extreme cases the average can even coincide with the arc's center. In 2D Euclidean space, this can be easily amended by averaging angles corresponding to the points $\mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{p}_3$ (see Figure 2 right).

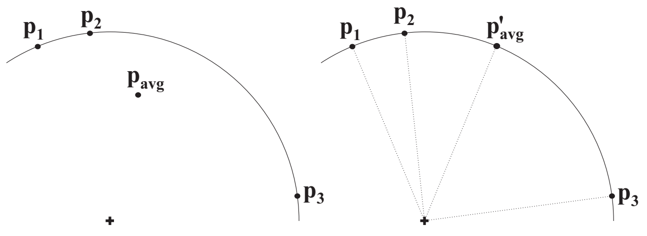


Figure 2: Averaging points (left) versus averaging angles (right).

The principle of our approach is the same as when blending normals, e.g., as in Phong shading (see Figure 3). When blending normals $\mathbf{n}_1$ and $\mathbf{n}_2$ with weights 0.3 and 0.7, we first blend them linearly, producing vector $\mathbf{n}_b$, and subsequently normalize it to the resulting normal $\mathbf{n}_{final}$. Even though this is not equivalent to the theoretically perfect intrinsic blending, the approximation is often sufficient (particularly when $\mathbf{n}_1$ and $\mathbf{n}_2$ are close) and the algorithm is very fast. Dual quaternions permit us to apply the same trick on $SE(3)$.

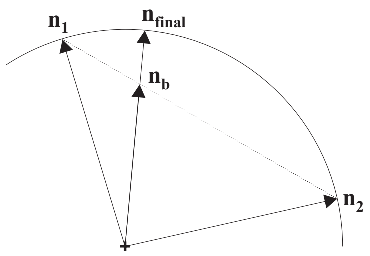


Figure 3: Standard normal blending trick: vectors $\mathbf{n}_1$ and $\mathbf{n}_2$ are first blended linearly giving $\mathbf{n}_b$ and then re-normalized, giving $\mathbf{n}_{final}$.

Moreover, the mathematical properties of dual quaternions ensure that none of the skin collapsing effects (see Section 2.2) exhibited by previous techniques will manifest themselves. Blending of dual quaternions can be elegantly computed in a vertex shader with complexity comparable to standard linear blending. Dual quaternions are more memory efficient, requiring only 8 floats per transformation (essentially, two regular quaternions), instead of the 12 required by matrices. In an existing application, it is extremely easy to replace a linear blend skinning implementation by a dual quaternion one. All that is necessary is a slight modification of the vertex shader and conversion of the matrices to dual quaternions before passing them to the shader. The model files as well as the internal data structures do not need any change at all – the only difference is in the transformation blending.

This paper extends the work presented in [Kavan et al. 2007]. We have added a more complete dual quaternion tutorial (Appendix A), providing proofs of all important statements and highlighting the connection to spatial kinematics. Addressing practical issues, we propose a more efficient vertex shader (Section 4), discuss quaternion antipodality issues (crucial for robust implementation, Section 4.1) and propose a method to integrate scale/shear joint transformations (Section 4.2).

Conventions. We denote scalars by lower-case letters, vectors, complex numbers and quaternions in 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 $\|\mathbf{v}\|$ is the usual vector norm. The cross product is denoted as $\mathbf{v} \times \mathbf{w}$ and takes precedence over vector addition, as is usual.

2 Related Work

The first reference to dual quaternions (historically called biquaternions) appears in [Clifford 1882]. More recent publications study dual quaternions from the viewpoint of theoretical kinematics [Bottema and Roth 1979; McCarthy 1990]. To date, dual quaternions have been applied mainly in computer vision and robotics [Daniilidis 1999; Perez and McCarthy 2004], but only rarely in computer graphics [Luciano and Banerjee 2000].

Geometric algebras. Dual quaternions are a special case of the more general concept of geometric algebras. These algebras naturally contain not only vectors and quaternions, but also k-dimensional subspaces [Wareham et al. 2005]. This leads to very elegant and dimension-independent expressions of geometric properties, but sometimes unfortunately also to an increase in time and memory complexity of the resulting implementation [Fontijne and Dorst 2003]. Dual quaternions, in turn, are not so general, but are more compact and faster to manipulate. For computer graphics practitioners, the big advantage of dual quaternions is that they are based on regular quaternions – a well-known tool in computer graphics [Shoemake 1985].

Blending vs interpolation. A vast amount of literature has been devoted to the problem of transformation interpolation [Barr et al. 1992; Juttler 1994; Marthinsen 1999; Belta and Kumar 2002; Hofer and Pottmann 2004; Li and Hao 2006; Wang et al. 2008]. This is not surprising, because the construction of interpolation curves for given key transformations (e.g., camera orientations) is a fundamental problem in computer animation. Unfortunately, in skinning, we face a different problem: the blending of rigid transformations, i.e. their weighted average (confusingly, in some literature this is also called *interpolation*). The weighted averages can be used to construct interpolation curves (see [Buss and Fillmore 2001]) but not vice versa.

2.1 Skinning

Historically, the idea of skin deformation by an underlying skeleton is credited to [Magnenat-Thalmann et al. 1988]. Since then, several different approaches to skeletal animation have emerged. Our approach – dual quaternion skinning – falls into the category of geometric methods. For completeness, however, we also survey other related techniques.

Physically based methods. A logical approach to character animation is to simulate the internal structure of the body: bones, muscles and fat tissues. This can work either with explicit anatomy knowledge [Scheepers et al. 1997; Aubel and Thalmann 2000; Teran et al. 2005], or without [Capell et al. 2002; Guo and Wong 2005; Pratscher et al. 2005]. Physically based methods generally obtain a high level of realism (delivering also dynamic effects and muscle bulges), but at high computational costs.

Capturing real subjects. Several methods successfully exploit modern motion capture and/or 3D scanning devices to capture skin deformation of real people [Allen et al. 2002; Anguelov et al. 2005; Park and Hodgins 2006; Allen et al. 2006]. These approaches are highly accurate, but require expensive hardware and are of course limited to existing subjects only.

Example based techniques. Multiple input meshes can be used both to resolve the artifacts of linear blending and to add additional effects like muscle bulging. Example based methods use either direct interpolation between example meshes [Lewis et al. 2000; Sloan et al. 2001], approximation by principal components of example deformations [Kry et al. 2002] or fit the linear blending parameters to match the provided examples [Mohr and Gleicher 2003]. A more accurate (yet more complex) example interpolation method has been proposed [Kurihara and Miyata 2004] and augmented with an innovative GPU approach [Rhee et al. 2006]. Recently, example based skinning methods have been improved by using rotational instead of linear regression [Wang et al. 2007; Weber et al. 2007]. Generally, this class of algorithms offers a level of realism limited only by the number of input examples. However, the production of examples can be costly, requiring a lot of artist labour.

Another possible way to overcome the limitations of linear blending with the aid of examples is to use more than one weight per matrix, resulting in a method called Multi-Weight Enveloping [Wang and Phillips 2002]. This idea has recently been refined by Merry et al., whose system is called Animation Space [2006]. The great advantage of Animation Space is that it is a linear framework, i.e., that blending is done in a linear space (albeit multi-dimensional) and yet it still significantly outperforms both linear blend skinning as well as multi-weight enveloping in terms of deformation quality [Jacka et al. 2007].

Geometric methods. In this case, only one input mesh is provided (designed in a reference pose). The skeleton-to-skin binding is defined in a direct, geometrical way. The most popular method, established with linear blend skinning, is to bind each vertex to one or more joints. In the latter case, the weight (amount of influence) of all influencing matrices must be specified. This weighting, as well as skeleton fitting, is typically done manually. However, an automatic procedure has been described recently [Baran and Popović 2007], thus simplifying the rigging process considerably.

Advanced blending methods, e.g., direct quaternion blending [Hejl 2004], log-matrix blending [Cordier and Magnenat-Thalmann 2005] and spherical blending [Kavan and Žára 2005] use the same rigging structure as linear blend skinning. Even though these techniques remove some of the artifacts, they still fall short of delivering natural skin deformation in all postures (see Figure 1 and Section 2.2).

Alternative rigging. Some researchers propose combatting skinning artifacts by implementing a different rigging method, for example based on swept surfaces [Hyun et al. 2005] or auxiliary curved skeletons [Yang et al. 2006; Forstmann and Ohya 2006; Forstmann et al. 2007]. In some cases, this also allows advanced effects to be animated, such as muscle bulging and skin creasing. The disadvantages include complexity of the GPU implementation (even though the latest method from Forstmann et al. [2007] is almost as fast as linear blend skinning) and inconsistency with the established rigging pipeline: new rigging tools and data formats are needed. In this paper, we argue that the problems of linear blending do not stem from incorrect or insufficient rigging, but from incorrect blending. With dual quaternion skinning, it is therefore not necessary to either change the rigging structures or to update existing 3D models.

Another important class of methods is based on generalization of barycentric coordinates [Ju et al. 2005; Joshi et al. 2007]. This enables the user to deform a character using a simpler mesh (deformation cage). This approach is very appealing mainly in off-line production, where high quality and direct control of the deformations are more important than run-time efficiency.

2.2 Geometric Skinning

In this section, we elaborate on geometric skinning methods with the rigging structure adopted from linear blend skinning (which is the de facto standard in the videogames industry). A 3D object conforming to this standard consists of skin, a skeleton and vertex weights. The skin is a 3D triangular mesh with no assumed topology or connectivity and the skeleton is a rooted tree (both are designed in a reference pose). The nodes of the skeleton represent joints and the edges can be interpreted as bones. However, each bone can be easily identified by its origin, so the difference between joints and bones is rather moot in our case (and in the literature, these terms are often used interchangeably; we will use the term joint). The transformations relating joints in the hierarchy are assumed to be rigid (until Section 4.2, where we propose a method to integrate scale/shear joint transformations). The vertex weights describe the skin-to-skeleton binding, i.e., the amount of influence of individual joints on each vertex.

Let us assume that there are p joints in our model. In the rest-pose, each joint has an associated local coordinate system. The transformation from the rest-pose of joint $j \in \{1, \dots, p\}$ to its actual position in the animated posture can be expressed by a rigid transformation matrix – let us denote this matrix as $C_j \in SE(3)$.

We assume that vertex $\mathbf{v}$ is attached to joints $j_1, \dots, j_n$ with weights $\mathbf{w} = (w_1, \dots, w_n)$. The indices $j_1, \dots, j_n$ are integers referring to the joints that influence a given vertex – in other words, they are indices into the array of joints. There is usually a fixed upper bound on n (the number of influencing joints), typically 4, due to graphics hardware considerations. The weights are normally assumed to be convex, i.e., $\sum_{i=1}^{n} w_i = 1$ and $w_i \geq 0$. However, this non-negativity is not exploited in our algorithms (analogously to linear blend skinning), so artists can feel free to experiment with negative vertex weights. The weight w_i represents the influence of joint j_i on vertex $\mathbf{v}$.

The vertex position in the mesh deformed by linear blend skinning is then computed as

$$\mathbf{v}' = \sum_{i=1}^{n} w_i C_{j_i} \mathbf{v} \tag{1}$$

that is, transforming vertex $\mathbf{v}$ by all influencing joint transformations C_{j_i} and taking a weighted average. This is reminiscent of Figure 2 left, which suggests why artifacts such as the "candy-wrapper" occur with linear blend skinning. To explain this artifact, consider a very simple arm rig with only two joints: j_1 corresponding to the shoulder and j_2 to the elbow joint (see Figure 4 left). Vertex $\mathbf{v}$ in the figure is equally influenced by both joints, i.e, $w_1 = w_2 = 0.5$, in order to achieve smooth skinning. Let us further assume that the arm is animated by twisting joint j_2 by 180 degrees around the x-axis. The joint transformations therefore can be written as

$$C_{j_1} = \begin{pmatrix} I & 0 \\ 0 & 1 \end{pmatrix}, \; C_{j_2} = \begin{pmatrix} R_x(180^\circ) & 0 \\ 0 & 1 \end{pmatrix}$$

where I denotes the 3×3 identity matrix and R_x denotes rotation about the x-axis. We see that averaging $C_{j_1}\mathbf{v}$ and $C_{j_2}\mathbf{v}$ produces vertex $\mathbf{v}'$ exactly at the position of joint j_2, i.e., the skin collapses to a single point. Examples of this effect with a realistic 3D model are shown in Figure 14 left.

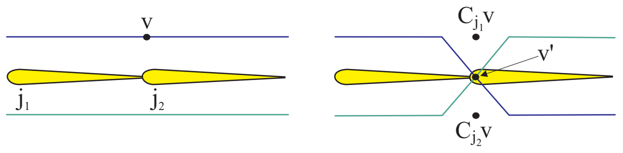


Figure 4: A typical "candy-wrapper" artifact of linear blend skinning. On the left is a reference pose and on the right an animated one.

To gain a better insight into linear blend skinning, we can re-write Equation (1) using distributivity of matrix-vector multiplication

$$\sum_{i=1}^{n} w_i C_{j_i} \mathbf{v} = \left(\sum_{i=1}^{n} w_i C_{j_i} \right) \mathbf{v} \tag{2}$$

The right-hand side of Equation (2) shows that skinning can also be computed by blending the transformations C_{j_i} first and then applying the result to $\mathbf{v}$, thereby obtaining the result in one step.

The transformation blending used in the right-hand side of Equation (2) is a direct linear combination of matrices. It is well known that this method is troublesome, because the blended matrix $\sum_{i=1}^{n} w_i C_{j_i}$ is not necessarily a rigid transformation, even if all C_{j_i} are rigid (i.e., the set of orthonormal matrices is not closed under addition). This is another way of explaining the linear blend skinning problems – undesired scaling creeps into our transformation matrix. In the extreme case shown in Figure 4, the blended matrix can even become rank deficient.

An obvious idea to improve skinning quality is to replace the linear blending of matrices in Equation (2) with a more sophisticated matrix blending technique. In particular, if the new blending technique will always produce a rigid transformation, we can expect that no candy-wrapper-like artifacts will occur. An ideal way would be to apply one of the $SE(3)$ intrinsic methods [Govindu 2004; Kavan et al. 2006]. This removes the artifacts of linear blend skinning but at a significant cost. The intrinsic methods are iterative and each iteration requires non-trivial computations (for details please refer to Appendix B.1). As a result, this approach has not become popular in real-time skinning.

Ideally, we would like to achieve high-quality skin deformation but with computational complexity comparable to that of linear blend skinning. The applied matrix blending method does not have to be exact, but should correctly handle both the rotational and translational parts of the transformations $C_{j_1}, \ldots, C_{j_n}$. Handling the translational component is non-trivial, because it depends on the chosen coordinate system (i.e., center of rotation). Let us illustrate what happens if we simply interpolate the translation vectors linearly. For our demonstration, we use the example of a human arm bending at the elbow. We set the shoulder transformation C_{j_1} to the identity, while C_{j_2} corresponds to the elbow bend. With respect to the coordinate system of the elbow joint, the transformation matrices have a simple form

$$C_{j_1} = \begin{pmatrix} I & 0 \\ 0 & 1 \end{pmatrix}, \; C_{j_2} = \begin{pmatrix} R_z(\alpha) & 0 \\ 0 & 1 \end{pmatrix}$$

We assume that the rotational blending is a 2D interpolation of the angle (see Figure 2 right) and that interpolation of the translation vectors is linear. For illustrative purposes only, Figure 5 left shows the resulting transformation C_{blend} applied to the whole mesh instead of individual vertices. As the translational part of both C_{j_1} and C_{j_2} is zero, the blended matrix is simply

$$C_{blend}(t) = \begin{pmatrix} R_z(\alpha t) & 0 \\ 0 & 1 \end{pmatrix}$$

Let us examine what happens if we choose a different coordinate system in which to express our transformations, for example that associated with the shoulder joint (see Figure 5 right). If $\mathbf{u}$ is the translation between the shoulder and the elbow joint and $T_\mathbf{u}$ is the corresponding translation matrix, we can express the transformations with respect to the shoulder joint as

$$\begin{aligned} C'_{j_1} &= T_\mathbf{u} C_{j_1} T_\mathbf{u}^{-1} = \begin{pmatrix} I & 0 \\ 0 & 1 \end{pmatrix} \\ C'_{j_2} &= T_\mathbf{u} C_{j_2} T_\mathbf{u}^{-1} = \begin{pmatrix} R_z(\alpha) & \mathbf{u} - R_z(\alpha)\mathbf{u} \\ 0 & 1 \end{pmatrix} \end{aligned}$$

Note that while the matrix C'_{j_2} differs from C_{j_2}, they both represent the same transformation (as is obvious from the bottom row of Figure 5). If we now interpolate between C'_{j_1} and C'_{j_2} using the same method as before, we obtain

$$C'_{blend}(t) = \begin{pmatrix} R_z(\alpha t) & t(\mathbf{u} - R_z(\alpha)\mathbf{u}) \\ 0 & 1 \end{pmatrix}$$

This is unfortunate because $C'_{blend}(t)$ no longer represents the same transformation as $C_{blend}(t)$ for $0 < t < 1$. To see this, compare the transformations of the elbow joint position by $C_{blend}(t)$ and $C'_{blend}(t)$. While the former leaves the elbow joint fixed, the latter produces the following trajectory

$$\mathbf{u}'(t) = C'_{blend}(t) \begin{pmatrix} \mathbf{u} \\ 1 \end{pmatrix} = R_z(\alpha t)\mathbf{u} - tR_z(\alpha)\mathbf{u} + t\mathbf{u}$$

For example, for $\alpha = 120^\circ$ and $\mathbf{u} = (2,0,0)$ (as in Figure 5) we obtain $\mathbf{u}'(0.5) = (2.5, \sqrt{3}/2, 0)$. This represents an unwanted drift away from the desired elbow position $\mathbf{u}$, as illustrated in Figure 5 right. This has, of course, catastrophical consequences for skinning. In practice, the situation is even worse, as the origin (i.e., the default center of rotation) is usually even further away (typically near the character's center of mass). See Figure 6 for an example with a 3D character model.

2.3 Center of Rotation Selection

The solution of the problem described in the previous section is straightforward, i.e., it is sufficient to set the rotation center to be in the appropriate joint. This is the basic idea of Hejl's method [2004], which assumes that the rotation center of vertex $\mathbf{v}$ is fixed and coincides with the joint nearest to vertex $\mathbf{v}$. For blending the rotational component, Hejl proposes to apply a linear combination of regular quaternions. This trick is of a similar nature to that of normal averaging (see Figure 3), but is instead performed on the unit quaternion hypersphere. Even though this is just an approximation of rigorous spherical averages [Buss and Fillmore 2001], it is sufficiently accurate for skinning and can be efficiently implemented on graphics hardware.

In cases similar to that of the elbow (two bones connected by a joint), this works perfectly. However, for more complex joint influences, e.g., when more than two influencing joints are involved,

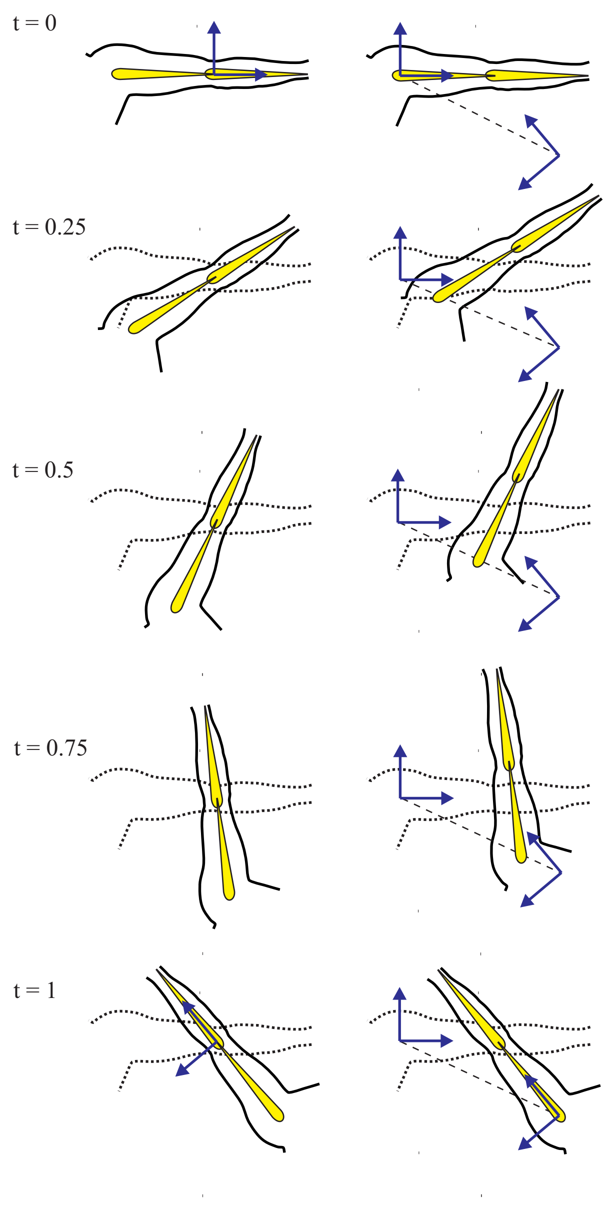


Figure 5: Interpolation of rigid transformations with rotation center in the elbow (left) and in the shoulder joint (right). The shoulder is not a suitable rotation center because it leads to an undesirable drift of the interpolated transformation.

selection of the center of rotation is not so simple. With character models, such difficult situations usually occur around the arm-pit or dorsum. To illustrate the problem more clearly, we use the example of a skeletally animated piece of cloth.

Let us assume that we have a fully outstretched piece of cloth influenced by two joints, as in Figure 7 left. Vertices close to the bone emanating from j_1 will follow transformation C_{j_1}, vertices close to the bone emanating from j_2 will follow transformation C_{j_2} and vertices in between will blend between these two transformations (to simulate the stretching effect). A problem occurs when switching the center of rotation from j_1 to j_2. For example, vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ in the figure are close to each other and therefore they will have similar vertex weights, i.e., approximately 0.5 for both j_1 and j_2. Therefore, both vertices will be rotated by approximately the same

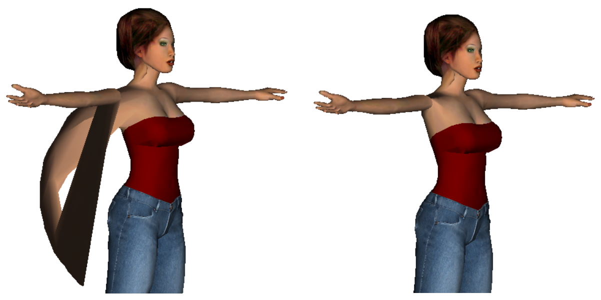


Figure 6: Artifacts produced by blending rotations with respect to the origin (left) are even worse than those of linear blend skinning (right).

amount (i.e., half of the rotation present in C_{j_1}). However, vertex $\mathbf{v}_1$ is slightly closer to j_1 and vertex $\mathbf{v}_2$ is slightly closer to j_2, which means that $\mathbf{v}_1$ will rotate about j_1, while $\mathbf{v}_2$ will rotate about j_2. This causes a problem depicted in Figure 7 right. Since the rotation centers of $\mathbf{v}_1$ and $\mathbf{v}_2$ differ considerably, the deformed vertices $\mathbf{v}'_1, \mathbf{v}'_2$ will be quite far apart. This is unfortunate, as nearby vertices $\mathbf{v}_1, \mathbf{v}_2$ in the reference skin should be mapped to nearby vertices $\mathbf{v}'_1, \mathbf{v}'_2$ in the deformed skin. Violation of this condition manifests itself as cracks, see Figure 15 left.

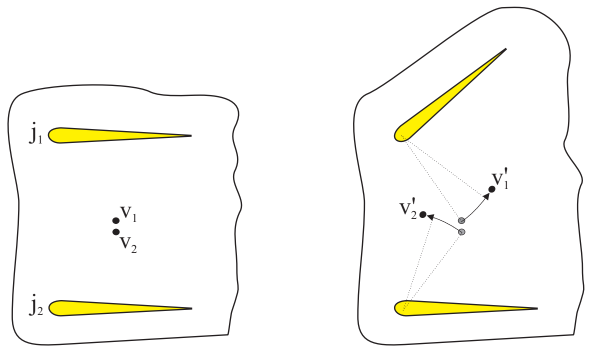


Figure 7: Problems with rotating vertices around their nearest joints. On the left is a reference pose and on the right an animated one.

A method to cope with these problems has been proposed by Kavan and Žára [2005]. This approach, called spherical blend skinning, also works by blending translations and rotations (represented by quaternions) independently. However, the center of rotation is selected in a more sophisticated way. In particular, the center of rotation is not fixed, but is computed at run-time using the actual joint transformations. To summarize, spherical blend skinning defines the center of rotation for vertex $\mathbf{v}$ influenced by joints $j_1, \ldots, j_n$ as the point $\mathbf{r}$ which minimizes

$$\sum_{1 \le a < b \le n} \|C_{j_a}\mathbf{r} - C_{j_b}\mathbf{r}\|$$

This is expressed as a least squares problem which is solved using Singular Value Decomposition (SVD).

Spherical blend skinning successfully handles situations such as that depicted in Figure 7, as both $\mathbf{v}_1$ and $\mathbf{v}_2$ are influenced by the same set of joints and therefore a suitable center of rotation will be computed. Unfortunately, spherical blend skinning is expensive in that the SVD algorithm is rather time consuming and thus it is not tractable to execute it once per vertex (if real-time speed is required). Therefore, spherical blend skinning uses the same rotation center for all vertices that are influenced by the same set of joints. Unfortunately, this can again cause discontinuous change of rotation centers, as demonstrated in Figure 8.

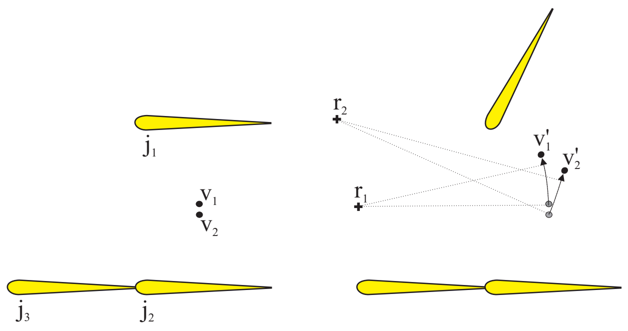


Figure 8: Problem with rotation centers in spherical blend skinning. On the left is a reference pose and on the right an animated one.

In this example, we modify the rigging from Figure 7 by adding one more joint, j_3. Let us assume $\mathbf{v}_2$ is influenced by all three joints (with j_3 having only a very small weight, e.g., $w_3 = 0.01$), while $\mathbf{v}_1$ is too far from j_3 and thus is influenced only by j_1 and j_2. Even though vertex $\mathbf{v}_2$ is only negligibly influenced by joint j_3, the algorithm computing rotation centers considers all influencing joints as equal (because the rotation center will be reused at other vertices). Therefore, the center of rotation $\mathbf{r}_1$ used to rotate vertex $\mathbf{v}_1$ will be different from the center of rotation $\mathbf{r}_2$ used to rotate $\mathbf{v}_2$, and we run into similar problems as before, though typically not so pronounced (see also Figure 17 left).

Intuitively, it is necessary to compute a proper center of rotation for every vertex individually, taking the joint weights into account. With spherical blend skinning, this is unfortunately not possible in real-time, as it would require execution of the SVD algorithm once per vertex. However, an interesting alternative would be to blend the rotation centers themselves. This idea has been investigated by Alexa [2002], using matrix exponentials and logarithms. The main principle is to linearly combine matrix logarithms instead of the matrices themselves (therefore the method is also known as *log-matrix blending*). The relationship between matrix logarithms and rotation centers might not be immediately obvious. However, note that the logarithm of a matrix $M \in SE(3)$ can be written as (see [Murray et al. 1994])

$$\log M = \begin{pmatrix} 0 & -\theta a_3 & \theta a_2 & m_1 \\ \theta a_3 & 0 & -\theta a_1 & m_2 \\ -\theta a_2 & \theta a_1 & 0 & m_3 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (3)$$

where θ is the angle of rotation, $\mathbf{a} = (a_1, a_2, a_3)$, $\|\mathbf{a}\| = 1$, is the direction of the rotation axis, $\mathbf{m} = (m_1, m_2, m_3) = \theta \mathbf{r} \times \mathbf{a} + d\mathbf{a}$, while $\mathbf{r}$ is the center of rotation and d is the *pitch* (ratio of translational and rotational velocity). Note that the center of rotation occurs in the cross product $\mathbf{r} \times \mathbf{a}$, which simply expresses the fact that any point $\mathbf{r} + t\mathbf{a}$ is also a valid rotation center (where $t \in R$ is arbitrary). All rotation centers therefore form a line in space, known also as the *screw* axis, see Appendix A.3.

Linear combination of matrix logarithms therefore involves linear combination of rotation centers. Log-matrix blending thus avoids the center of rotation problem inherent in quaternion-based methods, making it an appealing technique for skinning [Cordier and Magnenat-Thalmann 2005]. Unfortunately, log-matrix blending has another shortcoming, first pointed out in [Bloom et al. 2004]. The problem is that the matrix logarithm represents rotations in the so called *scaled axis* representation, i.e., represented by vector $\theta\mathbf{a}$, where θ is the angle of rotation and $\mathbf{a}$, $\|\mathbf{a}\| = 1$, is the direction of the axis of rotation. Interpolation of scaled rotation axes has certain undesirable properties, the most important being that it is not a shortest path interpolation. The reason is that the projection of a straight line onto a sphere (via exponential mapping) will not always result in a geodesic. However, geodesics correspond to shortest path interpolation – any deviation from the geodesic thus implies a longer than necessary trajectory (see Bloom et al. [2004] for details). In skinning, this manifests itself as an unnatural skin deformation (see Figure 16 left).

Note that it would be possible to improve this situation by selecting a different coordinate system with respect to which the matrices are expressed. This is possible for $n = 2$ and therefore also for piecewise linear curves [Li and Hao 2006]. However, it is unclear how to obtain an optimal coordinate system when blending $n > 2$ transformations. Nevertheless, this is an issue only with the scaled axis representation of rotations. With quaternions, we always obtain shortest path interpolation of rotations – even if the quaternions are blended linearly [Kavan and Žára 2005]. This is the reason why we do not encounter any non-shortest path artifacts with spherical blend skinning or with Hejl's method [2004].

3 Rigid Transformation Blending

From the discussion of previous geometric skinning methods, we see what an ideal rigid transformation blending method for skinning should look like. In particular, it should properly blend the centers of rotation (as with log-matrix blending), but also take advantage of quaternions to blend the rotational parts. A straightforward idea would be to simply blend the rotation centers linearly and couple them with rotations computed using quaternions. Unfortunately, as demonstrated below, this does not work. In this section, we will therefore discuss how the rotation centers should be blended properly.

Before we start, however, we need to prepare a formula expressing the center of rotation from a given rotation and translation (which will be useful not only in the following section, but also in Appendix A.3).

Lemma 1. *If a 2D rigid transformation is given by translation* $\mathbf{t} \in R^2$ *and angle of rotation* α*, then its center of rotation* $\mathbf{r}$ *is given as*

$$\mathbf{r} = \frac{1}{2}\left(\mathbf{t} + \mathbf{z} \times \mathbf{t} \operatorname{cotan}\frac{\alpha}{2}\right) \quad (4)$$

where $\mathbf{z} = (0, 0, 1)$ *is the z-axis.*

Proof. The formula can be derived elegantly using complex numbers – let us therefore assume that $\mathbf{t}$ is a complex number. The center of rotation being sought, $\mathbf{r} \in C$, is the stationary point of our rigid transformation, i.e.,

$$\mathbf{r} = \mathbf{t} + e^{i\alpha}\mathbf{r}$$

From this equation we can express the center of rotation as follows:

$$\mathbf{r} = \frac{\mathbf{t}}{1 - e^{i\alpha}} \cdot \frac{1 - e^{-i\alpha}}{1 - e^{-i\alpha}} = \frac{1 - e^{-i\alpha}}{2(1 - \cos\alpha)}\mathbf{t} = \frac{1 - \cos\alpha + i\sin\alpha}{2(1 - \cos\alpha)}\mathbf{t}$$

Using one of the well-known trigonometric identities

$$\frac{\sin\alpha}{1 - \cos\alpha} = \operatorname{cotan}\frac{\alpha}{2}$$

and the fact that multiplication by a complex unit is equivalent to a cross product with the z-axis, we obtain Equation (4). □

Note that the same formula applies in 3D, just using an arbitrary axis of rotation instead of the z-axis. This is because the translation component parallel to the rotation axis does not affect the center of rotation (see Appendix A.3).

3.1 2D Case

We will present a simple $SE(2)$ example which will illustrate the issues with blending centers of rotation. Let us assume we have two 2D rigid transformations $M_1, M_2 \in SE(2)$, where M_1 is given by angle α_1 and translation vector $(2\cos\alpha_1, 2\sin\alpha_1)$, and M_2 by angle α_2 and translation vector $(2\cos\alpha_2, 2\sin\alpha_2)$, see Figure 9.

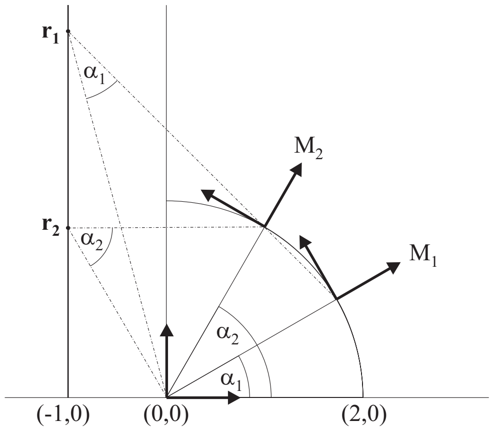


Figure 9: Interpolation between transformations M_1 and $M_2 \in SE(2)$ with rotation centers $\mathbf{r}_1$ and $\mathbf{r}_2$. Note that the angle of rotation is the same with respect to any center.

First of all, we need to express the rotation centers $\mathbf{r}_1$ and $\mathbf{r}_2$. In fact, we can easily find the rotation centers for all transformations from the family

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 2\cos\alpha \\ \sin\alpha & \cos\alpha & 2\sin\alpha \\ 0 & 0 & 1 \end{pmatrix}, \ \alpha \in \left[0, \frac{\pi}{2}\right]$$

If we plug $\mathbf{t} = (2\cos\alpha, 2\sin\alpha)$ into Equation (4), we obtain, after some simplifications,

$$\mathbf{r} = \left(-1,\ \frac{\sin\alpha}{1-\cos\alpha}\right) = \left(-1,\ \mathrm{cotan}\frac{\alpha}{2}\right) \tag{5}$$

The rotation center corresponding to M_i therefore is

$$\mathbf{r}_i = (r_{i,x},\ r_{i,y}) = \left(-1,\ \mathrm{cotan}\frac{\alpha_i}{2}\right),\ \ i = 1,2 \tag{6}$$

Now, we can turn our attention to interpolation between M_1 and M_2. Obviously, M_2 can be obtained from M_1 by composing M_1 with origin-centered rotation with angle $\alpha_2 - \alpha_1$. Therefore, the natural way to interpolate between M_1 and M_2 is along the spherical arc, depicted in Figure 9. An important question is what happens with the center of rotation of the interpolated transformation $M(t)$. Obviously, the rotation center of $M(t)$ will lie on the line segment determined by $\mathbf{r}_1$ and $\mathbf{r}_2$. However, from Equation (6), we see that the interpolation of rotation centers will not be linear. Let us test how far from the correct solution the linear combination of rotation centers is. The resulting trajectory for $\alpha_1 = 30$ and $\alpha_2 = 60$ degrees is shown in Figure 10. We see that it is quite far from the desired spherical arc, and therefore we can conclude that non-linear interpolation of rotation centers is indeed necessary.

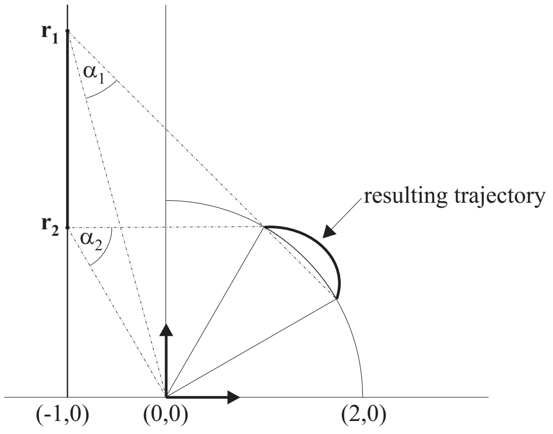


Figure 10: Trajectory resulting from linear interpolation of rotation centers.

Let us therefore derive the desired non-linear interpolant. First, however, a remark regarding interpolation of the angle: in 2D, it would be natural to simply interpolate the rotation angle linearly. Unfortunately, the corresponding 3D counterpart would involve proper spherical averages [Buss and Fillmore 2001], which is undesirable, because for fast skinning we would prefer linear quaternion blending. Therefore, to make our discussion relevant, we will consider the angle of rotation to be interpolated using the 2D version of linear quaternion blending. This amounts to interpolation between $(\cos\alpha_1, \sin\alpha_1)$ and $(\cos\alpha_2, \sin\alpha_2)$ along a straight line followed by projection back to the spherical arc, as shown in Figure 3. Moreover, to be fully compatible with quaternions, we will work with *half* of the rotation angle in trigonometric functions (noting that quaternions employ half of the angle of rotation).

Specifically, if we denote $\mathbf{p}_i = (\cos\frac{\alpha_i}{2}, \sin\frac{\alpha_i}{2})$, the interpolated angle $\alpha(t)$ will be given by

$$\left(\cos\frac{\alpha(t)}{2}, \sin\frac{\alpha(t)}{2}\right) = \frac{(1-t)\mathbf{p}_1 + t\mathbf{p}_2}{\|(1-t)\mathbf{p}_1 + t\mathbf{p}_2\|} \tag{7}$$

According to Equation (5), all we need, to compute the center of rotation corresponding to $\alpha(t)$, is $\sin\alpha(t)$ and $\cos\alpha(t)$. This can be retrieved from Equation (7) using the well known identities

$$\begin{aligned} \cos\alpha(t) &= \left(\cos\frac{\alpha(t)}{2}\right)^2 - \left(\sin\frac{\alpha(t)}{2}\right)^2 \\ \sin\alpha(t) &= 2\sin\frac{\alpha(t)}{2}\cos\frac{\alpha(t)}{2} \end{aligned}$$

The actual derivations are somewhat lengthy, therefore we employed Maple [Char et al. 1983] to perform all of the substitutions and simplifications. According to the listing (see Figure 11), the resulting formula for the y-coordinate of the rotation center is given by

$$r_y(t) = \frac{(1-t)\cos\frac{\alpha_1}{2} + t\cos\frac{\alpha_2}{2}}{(1-t)\sin\frac{\alpha_1}{2} + t\sin\frac{\alpha_2}{2}} \tag{8}$$

(for the x-coordinate we of course have $r_x(t) = -1$). Equation (8) can be re-written to

$$r_y(t) = \frac{(1-t)\sin\frac{\alpha_1}{2}\mathrm{cotan}\frac{\alpha_1}{2} + t\sin\frac{\alpha_2}{2}\mathrm{cotan}\frac{\alpha_2}{2}}{(1-t)\sin\frac{\alpha_1}{2} + t\sin\frac{\alpha_2}{2}}$$

```
>  p1x := cos(alpha1/2):  p1y := sin(alpha1/2):
>  p2x := cos(alpha2/2):  p2y := sin(alpha2/2):
>  ptx := (1-t)*p1x + t*p2x:  pty := (1-t)*p1y
+ t*p2y:
>  norm := sqrt(ptx^2 + pty^2):
>  cosAlphaThalf := ((1-t)*p1x + t*p2x) / norm:
>  sinAlphaThalf := ((1-t)*p1y + t*p2y) / norm:
>  cosAlphaT := simplify(cosAlphaThalf^2 -
sinAlphaThalf^2);
```

$$\begin{aligned}&cosAlphaT := -(2\cos(\tfrac{1}{2}\alpha 1)^2 - 4\cos(\tfrac{1}{2}\alpha 1)^2 t +\\ &2\cos(\tfrac{1}{2}\alpha 1)t\cos(\tfrac{1}{2}\alpha 2) + 2\cos(\tfrac{1}{2}\alpha 1)^2 t^2 -\\ &2\cos(\tfrac{1}{2}\alpha 1)t^2\cos(\tfrac{1}{2}\alpha 2) + 2t^2\cos(\tfrac{1}{2}\alpha 2)^2 - 1 + 2t -\\ &2\sin(\tfrac{1}{2}\alpha 1)t\sin(\tfrac{1}{2}\alpha 2) - 2t^2 + 2\sin(\tfrac{1}{2}\alpha 1)t^2\sin(\tfrac{1}{2}\alpha 2)) \Big/\\ &(-2\cos(\tfrac{1}{2}\alpha 1)t\cos(\tfrac{1}{2}\alpha 2) + 2\cos(\tfrac{1}{2}\alpha 1)t^2\cos(\tfrac{1}{2}\alpha 2) - 1 +\\ &2t - 2\sin(\tfrac{1}{2}\alpha 1)t\sin(\tfrac{1}{2}\alpha 2) - 2t^2 + 2\sin(\tfrac{1}{2}\alpha 1)t^2\sin(\tfrac{1}{2}\alpha 2))\end{aligned}$$

```
>  sinAlphaT := simplify(2*sinAlphaThalf*
cosAlphaThalf);
```

$$\begin{aligned}&sinAlphaT := -2((-\sin(\tfrac{1}{2}\alpha 1) + \sin(\tfrac{1}{2}\alpha 1)t - t\sin(\tfrac{1}{2}\alpha 2))\\ &(-\cos(\tfrac{1}{2}\alpha 1) + \cos(\tfrac{1}{2}\alpha 1)t - t\cos(\tfrac{1}{2}\alpha 2)))\Big/\\ &(-2\cos(\tfrac{1}{2}\alpha 1)t\cos(\tfrac{1}{2}\alpha 2) + 2\cos(\tfrac{1}{2}\alpha 1)t^2\cos(\tfrac{1}{2}\alpha 2) - 1 +\\ &2t - 2\sin(\tfrac{1}{2}\alpha 1)t\sin(\tfrac{1}{2}\alpha 2) - 2t^2 + 2\sin(\tfrac{1}{2}\alpha 1)t^2\sin(\tfrac{1}{2}\alpha 2))\end{aligned}$$

```
>  ryT := simplify(sinAlphaT/(1-cosAlphaT));
```

$$\begin{aligned}&ryT := -((-\sin(\tfrac{1}{2}\alpha 1) + \sin(\tfrac{1}{2}\alpha 1)t - t\sin(\tfrac{1}{2}\alpha 2))\\ &(-\cos(\tfrac{1}{2}\alpha 1) + \cos(\tfrac{1}{2}\alpha 1)t - t\cos(\tfrac{1}{2}\alpha 2)))\Big/\\ &(-1 + 2t - 2\sin(\tfrac{1}{2}\alpha 1)t\sin(\tfrac{1}{2}\alpha 2) - 2t^2 +\\ &2\sin(\tfrac{1}{2}\alpha 1)t^2\sin(\tfrac{1}{2}\alpha 2) + \cos(\tfrac{1}{2}\alpha 1)^2 - 2\cos(\tfrac{1}{2}\alpha 1)^2 t +\\ &\cos(\tfrac{1}{2}\alpha 1)^2 t^2 + t^2\cos(\tfrac{1}{2}\alpha 2)^2)\end{aligned}$$

```
>  simplify(denom(ryT)+(-sin(1/2*alpha1)+
sin(1/2*alpha1)*t-t*sin(1/2*alpha2))^2);
```

$$0$$

```
>  ryTfinal := (cos(1/2*alpha1)-cos(1/2*alpha1)*t+
t*cos(1/2*alpha2)) / (sin(1/2*alpha1)-
sin(1/2*alpha1)*t+t*sin(1/2*alpha2));
```

$$ryTfinal := \frac{\cos(\frac{1}{2}\alpha 1) - \cos(\frac{1}{2}\alpha 1)t + t\cos(\frac{1}{2}\alpha 2)}{\sin(\frac{1}{2}\alpha 1) - \sin(\frac{1}{2}\alpha 1)t + t\sin(\frac{1}{2}\alpha 2)}$$

```
>  simplify(ryTfinal - ryT);
```

$$0$$

Figure 11: Center of rotation derivation in Maple

$$= \frac{(1-t)\sin\frac{\alpha_1}{2}r_{1,y} + t\sin\frac{\alpha_2}{2}r_{2,y}}{(1-t)\sin\frac{\alpha_1}{2} + t\sin\frac{\alpha_2}{2}}$$

This allows us to conclude that when the angle of rotation is interpolated using linear quaternion blending, the corresponding center of rotation $\mathbf{r}(t)$ is interpolated non-linearly according to the formula

$$\mathbf{r}(t) = \frac{(1-t)\sin\frac{\alpha_1}{2}\mathbf{r}_1 + t\sin\frac{\alpha_2}{2}\mathbf{r}_2}{(1-t)\sin\frac{\alpha_1}{2} + t\sin\frac{\alpha_2}{2}} \tag{9}$$

This is, in fact, linear interpolation weighted by the factor $\sin\frac{\alpha_i}{2}$. This suggests that we could store the rotation center in the form of $\mathbf{r}_i\sin\frac{\alpha_i}{2}$ instead of just $\mathbf{r}_i$. Actually, if we add a unit quaternion to represent the rotation, we obtain a representation of $SE(2)$ that can be interpolated linearly without introducing discrepancy between the rotation and its center.

Have we discovered anything new? The answer is no. In fact, this representation of $SE(2)$ was first discovered in the 19th century [Clifford 1882] and is known today as *planar dual quaternions* [McCarthy 1990]. In particular, any planar dual quaternion $\hat{\mathbf{q}}$ has the form

$$\hat{\mathbf{q}} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}(k + \varepsilon i r_y - \varepsilon j r_x) \tag{10}$$

where θ is the angle of rotation and (r_x, r_y) is the center of rotation. Note that i, j, k are the usual (Hamilton's) quaternion units, while ε is the dual unit, i.e., a number with property $\varepsilon^2 = 0$ (the intuition is that ε is so small that its square vanishes completely). In the following, we will assume that the reader is familiar with basic dual quaternion operations and their properties – otherwise please see the enclosed tutorial (Appendix A).

3.2 3D Case

Using dual quaternions, the interpolation derived in Section 3.1 can be written very concisely. We call this method Dual quaternion Linear Blending (DLB) and define it as follows

$$DLB(t; \hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2) = \frac{(1-t)\hat{\mathbf{q}}_1 + t\hat{\mathbf{q}}_2}{\|(1-t)\hat{\mathbf{q}}_1 + t\hat{\mathbf{q}}_2\|}$$

where $\hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2$ are unit dual quaternions representing the input transformations. Note that if $\hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2$ are planar dual quaternions, the corresponding rotation centers exactly obey Equation (9). However, DLB is defined for general dual quaternions and therefore can be applied to arbitrary 3D rigid transformations.

What is the geometrical interpretation of the 3D DLB? Recall that a unit dual quaternion is a special representation of the screw parameters. In particular, as shown in Appendix A.3, any unit dual quaternion $\hat{\mathbf{q}}$ can be written as

$$\hat{\mathbf{q}} = \cos\frac{\theta_0}{2} + \mathbf{s}_0\sin\frac{\theta_0}{2} + \varepsilon\left(\mathbf{s}_\varepsilon\sin\frac{\theta_0}{2} - \frac{\theta_\varepsilon}{2}\sin\frac{\theta_0}{2} + \mathbf{s}_0\frac{\theta_\varepsilon}{2}\cos\frac{\theta_0}{2}\right)$$

where θ_0 is the angle of rotation, $\mathbf{s}_0$ is the direction of the axis of rotation, θ_ε is the amount of translation along the rotation axis and $\mathbf{s}_\varepsilon = \mathbf{r}\times\mathbf{s}_0$ is the moment of the rotation axis (where $\mathbf{r}$ is the center of rotation). Again, as in Equation (3), we see that the center of rotation occurs in the cross product with the direction of the axis of rotation. Note that if $\mathbf{s}_0 = (0,0,1)$, the expression above reduces to Equation (10) (i.e., the 2D case is nothing more than the 3D case restricted to rotations about the z-axis).

The interpretation of the terms occuring in $\hat{\mathbf{q}}$ is as follows. Obviously, the non-dual part, i.e., $\cos\frac{\theta_0}{2} + \mathbf{s}_0\sin\frac{\theta_0}{2}$ is simply the regular quaternion representing the rotational component. Regarding the dual part, we see that the moment $\mathbf{s}_\varepsilon$ is again multiplied by $\sin\frac{\theta_0}{2}$, as was the case in 2D (Section 3.1). The term $\frac{\theta_\varepsilon}{2}\sin\frac{\theta_0}{2}$ is the "padding" which normalises $\hat{\mathbf{q}}$ (it is analogous to the $\cos\frac{\theta_0}{2}$ element of a regular quaternion). This is not surprising because a rigid transformation has only 6 degrees of freedom, whereas there are 8 elements in a dual quaternion. The last term, i.e., $\mathbf{s}_0\frac{\theta_\varepsilon}{2}$ represents translation along the axis of rotation. We see that in this term, the weighting factor happens to be $\cos\frac{\theta_0}{2}$ (instead of $\sin\frac{\theta_0}{2}$).

If required, DLB can be computed even without using dual quaternions. The principle is to convert the input matrices to the screw parameters (i.e., the screw axis, the angle of rotation and the amount of translation), and then linearly blend these parameters multiplied by the appropriate weighting functions (i.e., either $\sin\frac{\theta_0}{2}$ or $\cos\frac{\theta_0}{2}$). However, the practical value of this approach is questionable, because it is obviously less efficient than blending dual quaternions. In fact, an immediate optimization would be to precompute the trigonometric functions and their products, which would lead to nothing but disguised dual quaternion elements. For readers more interested in the constructive approach to dual quaternions and the related kinematic issues, we refer them to [McCarthy 1990; Bottema and Roth 1979].

A big advantage of DLB is that it works for more than two rigid transformations. If these transformations are expressed as unit dual quaternions $\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_n$ with convex weights $\mathbf{w} = (w_1,\ldots,w_n)$, the generalized DLB is simply

$$DLB(\mathbf{w};\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_n) = \frac{w_1\hat{\mathbf{q}}_1+\ldots+w_n\hat{\mathbf{q}}_n}{\|w_1\hat{\mathbf{q}}_1+\ldots+w_n\hat{\mathbf{q}}_n\|} \qquad (11)$$

This is very useful for skinning, where we often need to blend more than two joint transformations. However, note that *DLB* is only an approximation of rigorously defined weighted averages. The perfectly correct blending algorithm can be obtained by generalizing the methods presented in [Buss and Fillmore 2001], see Section 3.4 and Appendix B.1.

3.3 Distributivity and Coordinate Invariance

In Section 3.1, we have shown that the requirement of correct handling of rotation centers in $SE(2)$ leads to planar dual quaternions. This opens a question as to whether this is just a coincidence, or rather a consequence of some more fundamental property. In the following, we argue that the latter is the case and that the crucial property of dual quaternions is called *distributivity*. In the language of geometry, it corresponds to *coordinate invariance*.

Let us start by recalling the definitions. Distributivity (of multiplication with respect to addition) requires that for any dual quaternions $\hat{\mathbf{q}}_1,\hat{\mathbf{q}}_2,\hat{\mathbf{q}}_3$

$$\begin{aligned}(\hat{\mathbf{q}}_1+\hat{\mathbf{q}}_2)\hat{\mathbf{q}}_3 &= \hat{\mathbf{q}}_1\hat{\mathbf{q}}_3+\hat{\mathbf{q}}_2\hat{\mathbf{q}}_3\\ \hat{\mathbf{q}}_3(\hat{\mathbf{q}}_1+\hat{\mathbf{q}}_2) &= \hat{\mathbf{q}}_3\hat{\mathbf{q}}_1+\hat{\mathbf{q}}_3\hat{\mathbf{q}}_2\end{aligned}$$

Note that because of non-commutativity, we do indeed need both of these equations, sometimes referred to as *right* and *left* distributivity.

Coordinate invariance appears in many contexts in computer graphics. In this paper, we are concerned with coordinate invariance of transformation blending. Let us denote a general interpolation between two arbitrary transformations $N_1,N_2 \in SE(3)$ as $\Phi(t;N_1,N_2)$, where $t \in [0,1]$ is the interpolation parameter. Right-invariance requires Φ to satisfy

$$\forall T \in SE(3):\ \Phi(t;N_1,N_2)T = \Phi(t;N_1T,N_2T)$$

while left-invariance requires

$$\forall T \in SE(3):\ T\Phi(t;N_1,N_2) = \Phi(t;TN_1,TN_2)$$

If Φ satisfies both right- and left-invariance, we say it is *bi-invariant*. It is obvious that bi-invariance implies coordinate invariance, requiring

$$\forall T \in SE(3):\ T\Phi(t;N_1,N_2)T^{-1} = \Phi(t;TN_1T^{-1},TN_2T^{-1})$$

If the interpolation Φ is defined via linear combination, there is an immediate connection between distributivity and coordinate invariance. This is the case of DLB, defined in Section 3.2. It is easy to see that bi-invariance of DLB follows directly from distributivity of dual quaternions, as shown in the following lemma.

Lemma 2. *For any unit dual quaternions $\hat{\mathbf{p}},\hat{\mathbf{q}}_1,\hat{\mathbf{q}}_2$ and any interpolation parameter $t \in [0,1]$, both of the following equations are true*

$$\begin{aligned}DLB(t;\hat{\mathbf{q}}_1\hat{\mathbf{p}},\hat{\mathbf{q}}_2\hat{\mathbf{p}}) &= DLB(t;\hat{\mathbf{q}}_1,\hat{\mathbf{q}}_2)\hat{\mathbf{p}}\\ DLB(t;\hat{\mathbf{p}}\hat{\mathbf{q}}_1,\hat{\mathbf{p}}\hat{\mathbf{q}}_2) &= \hat{\mathbf{p}}DLB(t;\hat{\mathbf{q}}_1,\hat{\mathbf{q}}_2)\end{aligned}$$

Proof. For proof of the right-invariance it is sufficient to use the right-distributivity of dual quaternions which implies $\|(1-t)\hat{\mathbf{q}}_1\hat{\mathbf{p}} + t\hat{\mathbf{q}}_2\hat{\mathbf{p}}\| = \|((1-t)\hat{\mathbf{q}}_1+t\hat{\mathbf{q}}_2)\hat{\mathbf{p}}\| = \|(1-t)\hat{\mathbf{q}}_1+t\hat{\mathbf{q}}_2\|\|\hat{\mathbf{p}}\| = \|(1-t)\hat{\mathbf{q}}_1+t\hat{\mathbf{q}}_2\|$ (as $\hat{\mathbf{p}}$ is a unit dual quaternion). We can thus write

$$\begin{aligned}DLB(t;\hat{\mathbf{q}}_1\hat{\mathbf{p}},\hat{\mathbf{q}}_2\hat{\mathbf{p}}) &= \frac{(1-t)\hat{\mathbf{q}}_1\hat{\mathbf{p}}+t\hat{\mathbf{q}}_2\hat{\mathbf{p}}}{\|(1-t)\hat{\mathbf{q}}_1\hat{\mathbf{p}}+t\hat{\mathbf{q}}_2\hat{\mathbf{p}}\|} =\\ &= \frac{(1-t)\hat{\mathbf{q}}_1+t\hat{\mathbf{q}}_2}{\|(1-t)\hat{\mathbf{q}}_1+t\hat{\mathbf{q}}_2\|}\hat{\mathbf{p}} = DLB(t;\hat{\mathbf{q}}_1,\hat{\mathbf{q}}_2)\hat{\mathbf{p}}\end{aligned}$$

Proof of the left-invariance is a direct analogy of the proof above (using left-distributivity of dual quaternions). □

It may seem that bi-invariance and distributivity are just theoretical properties without any practical implications. However, this is not the case, as we now illustrate on the example from Figure 9. Let us assume that unit planar dual quaternions $\hat{\mathbf{q}}_1,\hat{\mathbf{q}}_2$ correspond to our transformations $M_1,M_2 \in SE(2)$. As discussed in Section 3.1, the natural interpolation between M_1 and M_2 is to consider the relative motion between M_1 and M_2, interpolate it, then compose the result with M_1. In our example, that relative transformation was the origin-centered rotation with angle $\alpha_2-\alpha_1$. In the language of dual quaternions, this can be written as

$$DLB(t;1,\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)\hat{\mathbf{q}}_1$$

because 1 corresponds to the identity and $\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*$ to the relative transformation (i.e., in our case, pure rotation). However, thanks to bi-invariance, we have

$$DLB(t;1,\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)\hat{\mathbf{q}}_1 = DLB(t;\hat{\mathbf{q}}_1,\hat{\mathbf{q}}_2) \qquad (12)$$

We have re-derived the result from Section 3.1, i.e., that linear blending of dual quaternions represents a plausible way to interpolate between M_1 and M_2. Another interpretation of Equation (12) is that we do not have to evaluate the relative motion (i.e., $\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*$) explicitly, because linear blending of dual quaternions automatically blends only that relative component.

Bi-invariance can also explain the problems with linear blending of $(quaternion, translation)$ pairs encountered in [Hejl 2004; Kavan and Žára 2005]. To see this, let us formally define an algebra over $(quaternion, translation)$ pairs with multiplication $\otimes$

corresponding to composition of transformations, i.e., for two $(\mathit{quaternion}, \mathit{translation})$ pairs $(\mathbf{q}_0, \mathbf{t}_0)$, $(\mathbf{q}_1, \mathbf{t}_1)$, this is defined as

$$(\mathbf{q}_0, \mathbf{t}_0) \otimes (\mathbf{q}_1, \mathbf{t}_1) = (\mathbf{q}_0\mathbf{q}_1, \mathbf{t}_0 + \mathbf{q}_0\mathbf{t}_1\mathbf{q}_0^*)$$

interpreting the 3D vector $\mathbf{t}_1$ as a quaternion with zero scalar part (as is usual). The right distributivity would require that

$$(\mathbf{q}_0 + \mathbf{q}_1, \mathbf{t}_0 + \mathbf{t}_1) \otimes (\mathbf{q}_2, \mathbf{t}_2) = (\mathbf{q}_0, \mathbf{t}_0) \otimes (\mathbf{q}_2, \mathbf{t}_2) + (\mathbf{q}_1, \mathbf{t}_1) \otimes (\mathbf{q}_2, \mathbf{t}_2)$$

Expanding the right hand side yields

$$\begin{aligned}(\mathbf{q}_0, \mathbf{t}_0) \otimes (\mathbf{q}_2, \mathbf{t}_2) + (\mathbf{q}_1, \mathbf{t}_1) \otimes (\mathbf{q}_2, \mathbf{t}_2) &= \\ = (\mathbf{q}_0\mathbf{q}_2 + \mathbf{q}_1\mathbf{q}_2, \mathbf{t}_0 + \mathbf{t}_1 + \mathbf{q}_0\mathbf{t}_2\mathbf{q}_0^* + \mathbf{q}_1\mathbf{t}_2\mathbf{q}_1^*)\end{aligned}$$

while the left hand side expands to

$$\begin{aligned}(\mathbf{q}_0 + \mathbf{q}_1, \mathbf{t}_0 + \mathbf{t}_1) \otimes (\mathbf{q}_2, \mathbf{t}_2) &= \\ = (\mathbf{q}_0\mathbf{q}_2 + \mathbf{q}_1\mathbf{q}_2, \mathbf{t}_0 + \mathbf{t}_1 + (\mathbf{q}_0 + \mathbf{q}_1)\mathbf{t}_2(\mathbf{q}_0 + \mathbf{q}_1)^*)\end{aligned}$$

It can be shown that the term $(\mathbf{q}_0 + \mathbf{q}_1)\mathbf{t}_2(\mathbf{q}_0 + \mathbf{q}_1)^*$ is not equivalent to $\mathbf{q}_0\mathbf{t}_2\mathbf{q}_0^* + \mathbf{q}_1\mathbf{t}_2\mathbf{q}_1^*$, which precludes right distributivity. This simply reflects the fact that blending $(\mathit{quaternion}, \mathit{translation})$ pairs rotates about the origin. Note that this might be advantageous in some settings (e.g., in rigid body physics, where we can define the origin to coincide with the center of mass). However, as demonstrated in Section 2.2, this is a complication in skinning.

As a final remark, we note that both linear blending of matrices and linear blending of matrix logarithms are coordinate invariant. The former follows from the distributivity of matrix multiplication, while the latter follows from the properties of the matrix exponential and logarithm (see [Moakher 2002])

$$\begin{aligned}\forall N, T \in SE(3): \quad & \exp(TNT^{-1}) = T\exp(N)T^{-1} \\ \forall N, T \in SE(3): \quad & \log(TNT^{-1}) = T\log(N)T^{-1}\end{aligned}$$

This explains why we did not encounter any rotation-center issues with linear blend skinning or with log-matrix blending.

3.4 Accuracy of Dual Quaternion Linear Blending

As argued in Section 3.3, DLB is a plausible method to interpolate rigid transformations. However, it is not perfect, as it is not a group-intrinsic method (i.e., it involves the "normal-interpolation" trick shown in Figure 3). In this section we discuss whether this will introduce artifacts when employing DLB in skinning. For clarity, we will start with the case of two transformations. The first step is to establish the perfectly correct blending method, i.e., one that respects the geometry of the underlying group (in our case $SE(3)$).

In the case of $SO(3)$, the theoretically perfect solution is Spherical Linear Interpolation (SLERP) [Shoemake 1985]. Recall that for two unit quaternions $\mathbf{q}_1, \mathbf{q}_2$ with parameter $t \in [0,1]$, the formula is $SLERP(t; \mathbf{q}_1, \mathbf{q}_2) = (\mathbf{q}_2\mathbf{q}_1^*)^t\mathbf{q}_1$ (assuming that $\langle \mathbf{q}_1, \mathbf{q}_2 \rangle \geq 0$; this can always be enforced by negating one of the quaternions). With the aid of dual quaternions, SLERP can be easily generalized to $SE(3)$. We call the resulting method Screw Linear Interpolation (ScLERP), for reasons that will soon become clear. For any two unit dual quaternions $\hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2$, it is given as $ScLERP(t; \hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2) = (\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)^t\hat{\mathbf{q}}_1$. What is its geometric interpretation?

We can see that $\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*$ is a unit dual quaternion, which represents the relative motion between $\hat{\mathbf{q}}_1$ and $\hat{\mathbf{q}}_2$. The power can be written as $(\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)^t = \cos(t\frac{\hat{\alpha}}{2}) + \hat{\mathbf{n}}\sin(t\frac{\hat{\alpha}}{2})$ for some dual angle $\hat{\alpha}$ and dual vector $\hat{\mathbf{n}}$ (see Appendix A.4). The dual vector $\hat{\mathbf{n}}$ represents the axis of the screw motion and 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 $(t\alpha_0)$ and the amount of translation $(t\alpha_\varepsilon)$. 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 $t\alpha_0$, as well as the amount of translation $t\alpha_\varepsilon$, vary linearly with respect to the interpolation parameter t. This means that ScLERP is a constant speed and shortest path interpolation, as could have been expected because it is a generalization of SLERP. As discussed in Section 3.3, another crucial property we expect from ScLERP is coordinate invariance, which we prove in the following lemma.

Lemma 3. *ScLERP is bi-invariant, that is for any unit dual quaternions $\hat{\mathbf{p}}, \hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2$ and any interpolation parameter $t \in [0,1]$, both of the following equations are true*

$$\begin{aligned}ScLERP(t; \hat{\mathbf{q}}_1\hat{\mathbf{p}}, \hat{\mathbf{q}}_2\hat{\mathbf{p}}) &= ScLERP(t; \hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2)\hat{\mathbf{p}} \\ ScLERP(t; \hat{\mathbf{p}}\hat{\mathbf{q}}_1, \hat{\mathbf{p}}\hat{\mathbf{q}}_2) &= \hat{\mathbf{p}}ScLERP(t; \hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2)\end{aligned}$$

Proof. The right-invariance is easy to show, because

$$\begin{aligned}ScLERP(t; \hat{\mathbf{q}}_1\hat{\mathbf{p}}, \hat{\mathbf{q}}_2\hat{\mathbf{p}}) &= (\hat{\mathbf{q}}_2\hat{\mathbf{p}}\hat{\mathbf{p}}^*\hat{\mathbf{q}}_1^*)^t\hat{\mathbf{q}}_1\hat{\mathbf{p}} = (\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)^t\hat{\mathbf{q}}_1\hat{\mathbf{p}} = \\ &= ScLERP(t; \hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2)\hat{\mathbf{p}}\end{aligned}$$

Proving the left-invariance is a little more tricky: $ScLERP(t; \hat{\mathbf{p}}\hat{\mathbf{q}}_1, \hat{\mathbf{p}}\hat{\mathbf{q}}_2) = (\hat{\mathbf{p}}\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\hat{\mathbf{p}}^*)^t\hat{\mathbf{p}}\hat{\mathbf{q}}_1$. It is now sufficient to show that $(\hat{\mathbf{p}}\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\hat{\mathbf{p}}^*)^t = \hat{\mathbf{p}}(\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)^t\hat{\mathbf{p}}^*$, because this gives us $(\hat{\mathbf{p}}\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\hat{\mathbf{p}}^*)^t\hat{\mathbf{p}}\hat{\mathbf{q}}_1 = \hat{\mathbf{p}}(\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)^t\hat{\mathbf{p}}^*\hat{\mathbf{p}}\hat{\mathbf{q}}_1 = \hat{\mathbf{p}}ScLERP(t; \hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2)$. However, the power can be written as $(\hat{\mathbf{p}}\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\hat{\mathbf{p}}^*)^t = \exp(t\log(\hat{\mathbf{p}}\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\hat{\mathbf{p}}^*))$. Thanks to Lemma 14, we can derive

$$\begin{aligned}\exp(t\log(\hat{\mathbf{p}}\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\hat{\mathbf{p}}^*)) &= \exp(t\hat{\mathbf{p}}\log(\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)\hat{\mathbf{p}}^*) = \\ &= \hat{\mathbf{p}}\exp(t\log(\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*))\hat{\mathbf{p}}^* = \hat{\mathbf{p}}(\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)^t\hat{\mathbf{p}}^*\end{aligned}$$

which concludes the proof. □

We therefore see that ScLERP is an interpolation with the same behavior as SLERP, but generalized to $SE(3)$. Now, we can use ScLERP as the gold standard to compare DLB against. As shown in Lemma 2 and Lemma 3, both methods are bi-invariant. Note that from this follows the most important feature, i.e., that the error will not be dependent on the choice of the coordinate systems (otherwise the error could be unbounded). In fact, we can exploit this common property of ScLERP and DLB to simplify their comparison.

Specifically, instead of comparing $DLB(t; \hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2)$ directly with $ScLERP(t; \hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2)$, we rewrite them as $DLB(t; 1, \hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)\hat{\mathbf{q}}_1$ and $ScLERP(t; 1, \hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)\hat{\mathbf{q}}_1$, which is correct because of right-invariance. Since $\hat{\mathbf{q}}_1$ is the same in both expressions, it is sufficient to compare just $DLB(t; 1, \hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)$ with $ScLERP(t; 1, \hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)$, which is an easier problem. As $\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*$ is a unit dual quaternion, it can be written as $\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^* = \cos\frac{\hat{\alpha}}{2} + \hat{\mathbf{n}}\sin\frac{\hat{\alpha}}{2}$. This enables us to derive

$$DLB(t; 1, \hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*) = \frac{1 - t + t\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*}{\|1 - t + t\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\|} = \frac{1 - t + t\cos(\frac{\hat{\alpha}}{2}) + \hat{\mathbf{n}}t\sin(\frac{\hat{\alpha}}{2})}{\|1 - t + t\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\|}$$

$$ScLERP(t; 1, \hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*) = (\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)^t = \cos\left(t\frac{\hat{\alpha}}{2}\right) + \hat{\mathbf{n}}\sin\left(t\frac{\hat{\alpha}}{2}\right)$$

from which we see that both DLB and ScLERP use the same, constant screw axis $\hat{\mathbf{n}}$.

Therefore, the one difference between DLB and ScLERP can only be in the motion along the screw axis, i.e., in the angle of rotation and amount of translation. Since $DLB(t; 1, \hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*)$ is a unit dual quaternion, we can also write it in the form

$$DLB(t; 1, \hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*) = \cos\frac{\hat{\beta}_t}{2} + \hat{\mathbf{n}}\sin\frac{\hat{\beta}_t}{2}$$

By considering only the scalar part of this equation, we see that

$$\cos\frac{\hat{\beta}_t}{2} = \frac{1-t+t\cos(\frac{\hat{\alpha}}{2})}{\|1-t+t\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\|} \tag{13}$$

It is possible to compute an upper bound on the difference between DLB and ScLERP by expressing the dual angle $\hat{\beta}_t$ from Equation (13) and comparing it with ScLERP's dual angle $\hat{\alpha}t$. This is not difficult but requires a lengthy mathematical analysis: we therefore employed Maple to carry out the computations (see Appendix B). The result is that the angles of rotation in DLB and ScLERP always differ by less than 8.15 degrees (which is in accordance with the results reported in [Kavan and Žára 2005] for the case of regular quaternions). The amount of translation always differs by less than 15.1% of the translation present in $\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*$. Note that these results are *upper* bounds – in practice, the difference is usually much smaller. Therefore, in applications such as skinning, the difference between DLB and ScLERP will most likely not be visible at all, although this would need to be verified with a perceptual study.

The same is true for $n > 2$, even though the error analysis in this more general case is not so simple. The problem is that the generalization of SLERP for $n > 2$ leads to spherical averages proposed in [Buss and Fillmore 2001]. Therefore, the gold standard of $SE(3)$ blending for more than two transformations involves generalizing Buss and Filmore's algorithms to unit dual quaternions (which is non-trivial because the set of unit dual quaternions is not a hypersphere). We discuss this in more detail in Appendix B.1.

4 Implementation Notes

Equation (11) is the key to fast and plausible skinning. In this section, we discuss the implementation issues on a typical heterogeneous architecture consisting of a CPU and GPU. The first step is to convert the skinning matrices $C_1,\ldots,C_p$ (where p is the total number of joints) to dual quaternions $\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_p$, unless our application works with them already. This will typically be done on a CPU and does not take long, because the conversion to a dual quaternion involves just one quaternion multiplication and the number of joints p is usually quite small. Let us recall a concise multiplication formula for regular quaternions. If two regular quaternions $\mathbf{q}_1, \mathbf{q}_2$ are written using their scalar and vector parts, i.e., $\mathbf{q}_1 = a_1 + \mathbf{r}_1$ and $\mathbf{q}_2 = a_2 + \mathbf{r}_2$, then their multiplication can be expressed as

$$\mathbf{q}_1\mathbf{q}_2 = a_1a_2 - \langle\mathbf{r}_1,\mathbf{r}_2\rangle + a_1\mathbf{r}_2 + a_2\mathbf{r}_1 + \mathbf{r}_1\times\mathbf{r}_2 \tag{14}$$

The resulting dual quaternions $\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_p$ are then sent to the GPU as uniform parameters, each represented by a 2×4 matrix.

The skin deformation, i.e., the DLB itself and the vertex and normal transformations take place in the vertex shader. In contrast to our previous approach [Kavan et al. 2007], we do not convert the resulting unit dual quaternion to an homogeneous matrix, but apply it directly to transform each vertex and normal (which results in shorter vertex shader code). The first step of $DLB(\mathbf{w};\hat{\mathbf{q}}_{j_1},\ldots,\hat{\mathbf{q}}_{j_n})$, i.e., the computation of the linear combination $\hat{\mathbf{b}} = \sum_{i=1}^n w_i\hat{\mathbf{q}}_{j_i}$, is straightforward, as it consists only of a per-component linear combination. However, the subsequent normalization, i.e., the computation of $\hat{\mathbf{b}}' = \hat{\mathbf{b}}/\|\hat{\mathbf{b}}\|$ can be optimized as follows.

If the non-dual and dual parts of $\hat{\mathbf{b}}$ are $\mathbf{b}_0$ and $\mathbf{b}_\varepsilon$, then the norm is $\|\hat{\mathbf{b}}\| = \|\mathbf{b}_0\| + \varepsilon\frac{\langle\mathbf{b}_0,\mathbf{b}_\varepsilon\rangle}{\|\mathbf{b}_0\|}$, according to Equation (22). The inverse is given by

$$\frac{1}{\|\hat{\mathbf{b}}\|} = \frac{1}{\|\mathbf{b}_0\|} - \varepsilon\frac{\langle\mathbf{b}_0,\mathbf{b}_\varepsilon\rangle}{\|\mathbf{b}_0\|^3}$$

according to Equation (18). Therefore,

$$\hat{\mathbf{b}}' = \frac{\hat{\mathbf{b}}}{\|\hat{\mathbf{b}}\|} = (\mathbf{b}_0 + \varepsilon\mathbf{b}_\varepsilon)\frac{1}{\|\hat{\mathbf{b}}\|} = \underbrace{\frac{\mathbf{b}_0}{\|\mathbf{b}_0\|}}_{\hat{\mathbf{b}}'_0} + \varepsilon\underbrace{\left(\frac{\mathbf{b}_\varepsilon}{\|\mathbf{b}_0\|} - \frac{\mathbf{b}_0\langle\mathbf{b}_0,\mathbf{b}_\varepsilon\rangle}{\|\mathbf{b}_0\|^3}\right)}_{\hat{\mathbf{b}}'_\varepsilon}$$

So, the rotational part of $\hat{\mathbf{b}}'$ is $\mathbf{b}'_0 = \frac{\mathbf{b}_0}{\|\mathbf{b}_0\|}$ and the translation is given by the vector part of $2\mathbf{b}'_\varepsilon(\mathbf{b}'_0)^*$. The latter expands to

$$2\mathbf{b}'_\varepsilon(\mathbf{b}'_0)^* = 2\left(\frac{\mathbf{b}_\varepsilon}{\|\mathbf{b}_0\|} - \frac{\mathbf{b}_0\langle\mathbf{b}_0,\mathbf{b}_\varepsilon\rangle}{\|\mathbf{b}_0\|^3}\right)\frac{\mathbf{b}_0^*}{\|\mathbf{b}_0\|} = 2\left(\frac{\mathbf{b}_\varepsilon\mathbf{b}_0^*}{\|\mathbf{b}_0\|^2} - \frac{\langle\mathbf{b}_0,\mathbf{b}_\varepsilon\rangle}{\|\mathbf{b}_0\|^2}\right)$$

Since the scalar part of $2\mathbf{b}'_\varepsilon(\mathbf{b}'_0)^* = 0$ (because $\hat{\mathbf{b}}'$ is unit), it means that there is no need to evaluate $\langle\mathbf{b}_0,\mathbf{b}_\varepsilon\rangle/\|\mathbf{b}_0\|^2$, because its purpose is only to cancel out the scalar part of $\mathbf{b}_\varepsilon\mathbf{b}_0^*/\|\mathbf{b}_0\|^2$. Therefore, we can compute the translational part of the matrix M just by computing the vector part of $2\mathbf{b}_\varepsilon\mathbf{b}_0^*/\|\mathbf{b}_0\|^2$ (and its scalar part can be safely ignored). This means that instead of computing the full dual quaternion normalization, all we need to compute is $\mathbf{c}_0 = \mathbf{b}_0/\|\mathbf{b}_0\|$ and $\mathbf{c}_\varepsilon = \mathbf{b}_\varepsilon/\|\mathbf{b}_0\|$ and retrieve the translation as the vector part of $2\mathbf{c}_\varepsilon\mathbf{c}_0^*$, which can be done efficiently using Equation (14).

The regular quaternion $\mathbf{c}_0$ is used to rotate the input vertex $\mathbf{v}$ and the normal $\mathbf{v}_n$. The key to a fast GPU implementation is the following classical formula [Shreiner et al. 2007], which describes how to efficiently express quaternion rotation in terms of cross products.

Lemma 4. *Let $\mathbf{q} = a + \mathbf{r}$ be a unit regular quaternion with scalar part a and vector part $\mathbf{r}$. Rotation of a vector (v_0, v_1, v_2) represented by the regular quaternion $\mathbf{v} = v_0 i + v_1 j + v_2 k$ can be computed as*

$$\mathbf{v}' = \mathbf{v} + 2\mathbf{r}\times(\mathbf{r}\times\mathbf{v} + a\mathbf{v}) \tag{15}$$

where $\mathbf{v}'$ is the vector $\mathbf{v}$ rotated by $\mathbf{q}$.

Proof. The proof consists of re-arranging the well-known expression $\mathbf{v}' = \mathbf{q}\mathbf{v}\mathbf{q}^*$. Using Equation (14), we can expand

$$\begin{aligned}\mathbf{v}' &= \mathbf{q}\mathbf{v}\mathbf{q}^* = (a+\mathbf{r})\mathbf{v}(a-\mathbf{r}) = (-\langle\mathbf{r},\mathbf{v}\rangle + a\mathbf{v} + \mathbf{r}\times\mathbf{v})(a-\mathbf{r}) = \\ &= -a\langle\mathbf{r},\mathbf{v}\rangle + a\langle\mathbf{v},\mathbf{r}\rangle + \langle\mathbf{r},\mathbf{v}\rangle\mathbf{r} + a^2\mathbf{v} + a(\mathbf{r}\times\mathbf{v}) - a(\mathbf{v}\times\mathbf{r}) - \\ &\quad (\mathbf{r}\times\mathbf{v})\times\mathbf{r} = \\ &= \langle\mathbf{r},\mathbf{v}\rangle\mathbf{r} + a^2\mathbf{v} + 2a(\mathbf{r}\times\mathbf{v}) + \mathbf{r}\times(\mathbf{r}\times\mathbf{v})\end{aligned}$$

Recall Lagrange's formula

$$\mathbf{r}\times(\mathbf{r}\times\mathbf{v}) = \mathbf{r}\langle\mathbf{r},\mathbf{v}\rangle - \mathbf{v}\langle\mathbf{r},\mathbf{r}\rangle$$

which, added to the previous equation, results in

$$\begin{aligned}\mathbf{v}' &= \langle\mathbf{r},\mathbf{v}\rangle\mathbf{r} + a^2\mathbf{v} + 2a(\mathbf{r}\times\mathbf{v}) + 2\mathbf{r}\times(\mathbf{r}\times\mathbf{v}) - \mathbf{r}\langle\mathbf{r},\mathbf{v}\rangle + \mathbf{v}\langle\mathbf{r},\mathbf{r}\rangle = \\ &= \mathbf{v}(a^2 + \|\mathbf{r}\|^2) + 2a(\mathbf{r}\times\mathbf{v}) + 2\mathbf{r}\times(\mathbf{r}\times\mathbf{v})\end{aligned}$$

from which Equation (15) readily follows. □

Note that the shader compiler could, in theory, perform the optimizations of $\mathbf{q}\mathbf{v}\mathbf{q}^*$ itself. However, according to our experiments, shader compilers produce suboptimal code when compared to implementation of Equation (15) (which translates to efficient code easily as modern GPUs provide fast cross product operations). An optimized dual quaternion skinning implementation can thus be summarized as Algorithm 1.

The resulting algorithm is encouragingly simple. However, actual vertex shader code would add some lighting computations and transformation by the model-view-projection matrix.

Algorithm 1

```
Input:  dual quaternions $\hat{\mathbf{q}}_1, \ldots, \hat{\mathbf{q}}_p$ (uniform parameters)
        vertex position $\mathbf{v}$ and normal $\mathbf{v}_n$
        joints indices $j_1, \ldots, j_n$ and weights $w_1, \ldots, w_n$
Output: transformed vertex position $\mathbf{v}'$ and normal $\mathbf{v}'_n$

  $\hat{\mathbf{b}} = w_1\hat{\mathbf{q}}_{j_1} + \ldots + w_n\hat{\mathbf{q}}_{j_n}$
  // denote the non-dual part of $\hat{\mathbf{b}}$ as $\mathbf{b}_0$ and the dual one as $\mathbf{b}_\varepsilon$
  $\mathbf{c}_0 = \mathbf{b}_0/\|\mathbf{b}_0\|$
  $\mathbf{c}_\varepsilon = \mathbf{b}_\varepsilon/\|\mathbf{b}_0\|$
  // denote the scalar part of $\mathbf{c}_0$ as $a_0$ and its vector part as $\mathbf{d}_0$
  // denote the scalar part of $\mathbf{c}_\varepsilon$ as $a_\varepsilon$ and its vector part as $\mathbf{d}_\varepsilon$
  $\mathbf{v}' = \mathbf{v} + 2\mathbf{d}_0 \times (\mathbf{d}_0 \times \mathbf{v} + a_0\mathbf{v}) + 2(a_0\mathbf{d}_\varepsilon - a_\varepsilon\mathbf{d}_0 + \mathbf{d}_0 \times \mathbf{d}_\varepsilon)$
  $\mathbf{v}'_n = \mathbf{v}_n + 2\mathbf{d}_0 \times (\mathbf{d}_0 \times \mathbf{v}_n + a_0\mathbf{v}_n)$
  // note that $\mathbf{v}'_n$ must be transformed by the inverse transpose matrix
```

4.1 Coping with Antipodality

Conversion of homogeneous matrices to dual quaternions is straightforward, up to the point where an appropriate sign for the resulting dual quaternion must be chosen. Due to dual quaternion antipodality, both $\hat{\mathbf{q}}_i$ and $-\hat{\mathbf{q}}_i$ represent the same rigid transformation, but their interpolation can be different (see Figure 22). The problem is exactly the same as for regular quaternions and occurs with all previous quaternion-based methods [Hejl 2004; Kavan and Žára 2005]. In the following, we restrict ourselves to the case of regular quaternions (as generalization to dual quaternions is trivial).

In the case of only two regular quaternions $\mathbf{q}_{j_1}, \mathbf{q}_{j_2}$ (i.e., in the case of classical SLERP [Shoemake 1985]), the situation is simple: we take either $\mathbf{q}_{j_2}$ or $-\mathbf{q}_{j_2}$, whichever gives a non-negative dot product with $\mathbf{q}_{j_1}$. In the general case ($n > 2$), we can by analogy require a non-negative dot product for all pairs of the resulting quaternions. More formally, we are looking for n numbers $s_1, \ldots, s_n \in \{0, 1\}$ such that the following holds

$$\forall h, i \in \{1, \ldots, n\} : \langle (-1)^{s_h}\mathbf{q}_{j_h}, (-1)^{s_i}\mathbf{q}_{j_i} \rangle \geq 0 \qquad (16)$$

A sequence of $s_1, \ldots, s_n$ will be called *valid* if it satisfies Equation (16). With dual quaternions, the only difference is that the dot product in Equation (16) is taken only between the non-dual parts of $\hat{\mathbf{q}}_{j_h}, \hat{\mathbf{q}}_{j_i}$ (to keep the notation simpler, we thus discuss only the regular quaternion case). A very simple algorithm to find $s_1, \ldots, s_n$ follows.

Algorithm 2

```
Input:  Regular quaternions $\mathbf{q}_{j_1}, \ldots, \mathbf{q}_{j_n}$
Output: Sequence $s_1, \ldots, s_n \in \{0, 1\}$

  $s_1 = 0$
  for $i = 2$ to $n$ do
    if $\langle \mathbf{q}_{j_1}, \mathbf{q}_{j_i} \rangle \geq 0$ then
      $s_i = 0$
    else
      $s_i = 1$
    end if
  end for
```

Note that Algorithm 2 is already commonly known, for example in the gaming community. In the light of the more sophisticated techniques of [Park et al. 2002; Johnson 2003], we initially conjectured that this algorithm provides just an approximate solution. However, as we show below, the algorithm actually always finds a valid sequence $s_1, \ldots, s_n$, if one exists.

Lemma 5. *If there exists a valid sequence $s_1, \ldots, s_n \in \{0, 1\}$ for regular quaternions $\mathbf{q}_{j_1}, \ldots, \mathbf{q}_{j_n}$, then Algorithm 2 finds one of the valid sequences.*

Proof. First of all, note that if $s'_1, \ldots, s'_n$ is valid, then so is $1 - s'_1, \ldots, 1 - s'_n$, as negating both quaternions does not change the sign of the dot product in Equation (16). Therefore, if a valid sequence exists then we can assume that the sequence may be given as $r_1, \ldots, r_n \in \{0, 1\}$ such that $r_1 = 0$. We show that our algorithm will return $s_i = r_i$ for $i = 1, \ldots, n$. This is obviously true for $i = 1$, as $s_1 = r_1 = 0$. For $i > 1$, we know that

$$\langle \mathbf{q}_{j_1}, (-1)^{r_i}\mathbf{q}_{j_i} \rangle \geq 0$$

because the sequence $r_1, \ldots, r_n$ is valid. The case of $r_i = 0$ means that $\langle \mathbf{q}_{j_1}, \mathbf{q}_{j_i} \rangle \geq 0$ and therefore our algorithm sets also $s_i = 0$. In the case of $r_i = 1$, we have $\langle \mathbf{q}_{j_1}, \mathbf{q}_{j_i} \rangle < 0$, so the 'else' branch in the if-statement is taken and the algorithm sets $s_i = 1$. □

In extreme situations, it is possible that no valid sequence exists. This means that for any choice of signs s_i, there will always exist a pair of rotations that will be interpolated along the longer path, see Figure 22. Luckily, such situations almost never occur in skinning.

Even though our antipodality resolution algorithm is quite simple and fast, it needs to be executed in the vertex shader, because it depends on the joint set $j_1, \ldots, j_n$ influencing a particular vertex. If a shorter vertex shader code is required, it is possible to resort to the following approximate method, which does not guarantee a valid sequence even if one exists. However, it allows the signs to be pre-computed before sending dual quaternions to the GPU, so the vertex shader will not have to care about antipodality. The idea is simple: we assume that each joint in the actual skeleton posture has rotated by less than 180 degrees with respect to its parent (i.e., has taken the shorter path). This assumption is practically always valid with character models, as joint rotations larger than 180 degrees are not normally possible. Therefore, we can set $s_1 = 0$ and select the remaining signs $s_2, \ldots, s_n$ so that

$$\forall i \in \{2, \ldots, n\} : \langle (-1)^{s_{\pi(i)}}\mathbf{q}_{\pi(i)}, (-1)^{s_i}\mathbf{q}_i \rangle \geq 0$$

where $\pi(i)$ denotes the parent joint of joint i. An algorithm to find these signs is a simple modification of Algorithm 2. Even though, theoretically, this algorithm does not necessarily produce valid signs for every vertex, we have not encountered any problems with our test data.

4.2 Non-rigid Joint Transformations

Dual quaternions cannot represent non-rigid transformations, such as scale and shear. This means that dual quaternion skinning, unlike linear blend skinning, is restricted only to rotating and/or translating joints. This does not seem prohibitive at first, given that any physical joints can also only rotate and/or translate. However, virtual characters can benefit from allowing non-rigid transformations. For example, non-uniform scaling can be applied to vary a character's proportions without having to modify the 3D model itself – a useful feature, for example in crowd animation. Some artists also use non-uniformly scaling joints to imitate muscle bulging and similar effects (which can even be done automatically as discussed by Mohr and Gleicher [2003]).

A natural way to support non-rigid transformations would be via higher-dimensional geometric algebras, which generalize dual

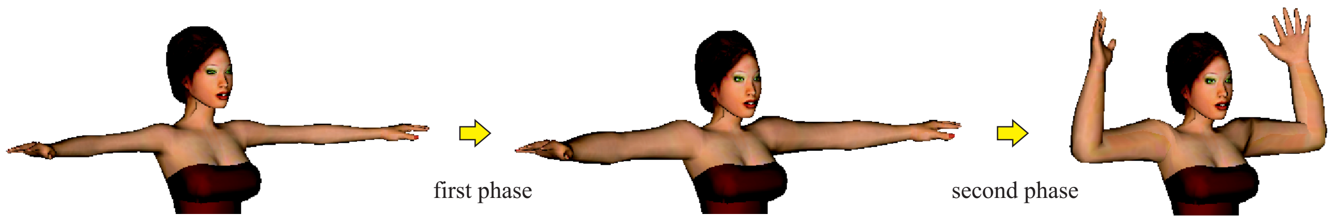


Figure 12: Adding scale transformations: in the first phase, we re-scale the mesh in the rest-pose. In the second phase, rigid joint transformations are applied using dual quaternion skinning.

quaternions. For example, 5D conformal geometric algebra [Wareham et al. 2005] supports rigid transformations along with a dilation (uniform scale). Unfortunately, to support the whole range of non-rigid transformations, including non-uniform scale and shear, one would have to move to even higher dimensional geometric algebras, suggesting a great cost (as an n-dimensional geometric algebra requires 2^n coordinates). Even though this approach would be theoretically interesting, for practical purposes we propose a simpler way of incorporating non-rigid transformations.

The idea is to separate the joint transformations into non-rigid and rigid parts and perform skinning in two phases. In the first phase, we deal with the non-rigid portion of the transformations, inflating the rest-pose mesh as required. In the second phase, we apply the rigid part to bend the mesh to the final shape. As no rotations are involved in the first pass, we can safely apply linear blending of transformation matrices [Shoemake and Duff 1992]. As only rigid transformations are involved in the second pass, we can apply dual quaternion skinning as described in Section 4. Composition of both steps yields the desired result, see Figure 12. Note that Kurihara and Nishita [2007] proposed a similar technique independently.

To decompose each skinning matrix C_i, $i = 1,\dots,p$ into a non-rigid and rigid part, we could apply polar decomposition [Shoemake and Duff 1992]. However, a more efficient solution is possible when considering how the matrices C_i are formed from individual joint transformations. Note that this would not work if no skeleton is present [James and Twigg 2005]. However, most character animation systems use skeletal animation. In the following we therefore assume we have a skeleton. Let us denote the transformation from the model coordinate system (frame) to joint i's frame in the rest-pose as A_i (absolute matrix) and the transformation from the model frame to joint i in the animated skeleton as F_i (final matrix). Then the skinning transformation C_i (composed matrix) can be written as $C_i = F_i(A_i)^{-1}$.

The absolute matrix A_i is simply a concatenation of individual joint transformations, i.e.,

$$A_i = R_1 \dots R_{\pi(i)} R_i$$

where $\pi(i)$ denotes the parent joint of joint i and R_i is a "relative matrix" describing the transformation from joint $\pi(i)$ to i in the rest-pose skeleton. Matrix R_1 describes the root joint transformation with respect to the model coordinate system (often $R_1 = I$, as the root is usually placed at the origin). Note that matrices R_i are assumed to be rigid.

The final matrix F_i is formed in a similar way, but accounting for the joint transformations $T_i \in SE(3)$

$$F_i = R_1 T_1 \dots R_{\pi(i)} T_{\pi(i)} R_i T_i$$

In order to enable non-rigid joint transformations, we could simply multiply T_i with a scale/shear matrix S_i. Even though our proposed method works in this case as well, we did not find it practical, because the scale/shear transformation S_i affects all descendants of joint i. For example, when scaling a spine joint (e.g., to obtain a larger belly), vertices of the arms get scaled in the same direction, which is typically not desirable. Therefore, we propose the scale/shear transformation S_i to be considered local, i.e., to affect only joint i's transformation. In our opinion, this results in more intuitive editing, as one usually wants to enlarge/thicken individual joints rather than the whole sub-tree.

According to the two-phase skinning paradigm introduced above, we define different matrices for each phase: A'_i, F'_i, C'_i for phase one, and A''_i, F''_i, C''_i for phase two. Concerning the first phase, $A'_i = R_1 \dots R_{\pi(i)} R_i$ as before, but

$$F'_i = R'_1 \dots R'_{\pi(i)} R'_i S_i$$

where $R'_k, k = 1,\dots,i$, is matrix R_k with its translational part scaled by $S_{\pi(k)}$ (rotational part being kept intact). This is to account for bone elongations caused by the non-rigid transformation present in the parent joint (note that $R'_1 = R_1$). The composed matrix for the first phase, $C'_i = F'_i(A'_i)^{-1}$, thus performs local scaling of joint i in the rest-pose (see Figure 12).

Concerning the second phase, the absolute matrix of joint i is

$$A''_i = R'_1 \dots R'_{\pi(i)} R'_i = F'_i S_i^{-1}$$

i.e., the final matrix of the first phase without the scale/shear factor S_i. The final matrix of the second phase deals with the joint rigid transformations T_i

$$F''_i = R'_1 T_1 \dots R'_{\pi(i)} T_{\pi(i)} R'_i T_i$$

and thus differs from the original final matrix F_i just by employing the elongated bone transformations R'_i. Therefore, the composed matrix of the second phase, $C''_i = F''_i(A''_i)^{-1}$, is rigid.

It is simple to modify the vertex shader code from Section 4 in order to implement this method. Matrices $C''_i, i = 1,\dots,p$ are converted to dual quaternions $\hat{\mathbf{q}}_i$, which are passed to the GPU along with matrices $C'_i, i = 1,\dots,p$. In the vertex shader, for a given vertex $\mathbf{v}$ associated with joints $j_1,\dots,j_n$ with weights $w_1,\dots,w_n$, we first transform the vertex by $\sum_{i=1}^n w_{j_i} C'_{j_i}$ and then by the blended dual quaternion as in Section 4. We deal with vertex normals in a similar way, just applying the inverse transpose of $\sum_{i=1}^n w_{j_i} C'_{j_i}$. The results of this method are shown in Figure 13, while the performance issues are discussed in the next section.

5 Results and Comparison

In our experiments, we use a human model with 5002 vertices, 9253 triangles and 54 joints. First, we consider only rigid joint

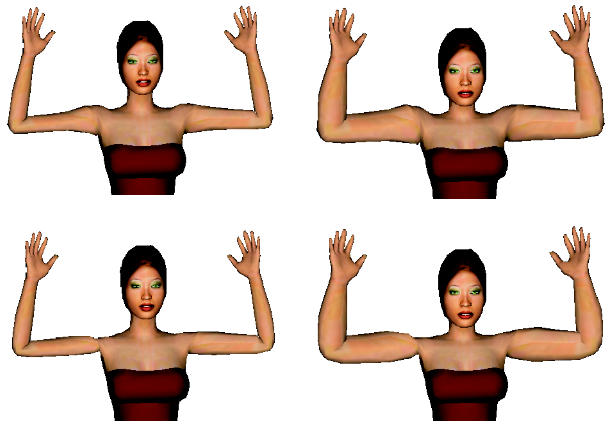


Figure 13: Proportions of the original model (left) are changed by applying non-uniform scaling to the character's arms (right). Our technique allows scaling transformations to be combined with dual quaternion skinning (top), thus achieving more realistic deformations than with linear blend skinning (bottom).

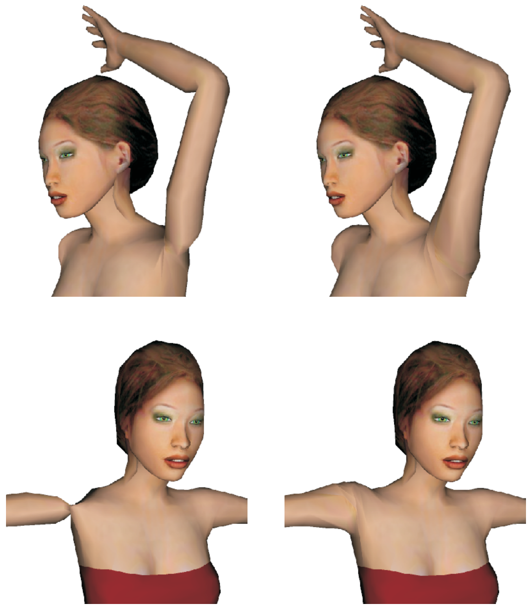


Figure 14: Comparison of linear (left) and dual quaternion (right) blending. Dual quaternions preserve rigidity of input transformations and therefore avoid skin collapsing artifacts.

transformations and compare the proposed dual quaternion skinning with linear blending, direct quaternion blending [Hejl 2004], log-matrix blending [Cordier and Magnenat-Thalmann 2005] and spherical blend skinning [Kavan and Žára 2005]. As discussed in Section 2.2, some artifacts are better visualized on a simple model of cloth (6000 vertices, 12000 triangles and 49 joints). Note that the only variable in our experiments is the transformation blending – the input data (model files and postures) are always the same. The visual results confirm that our DLB method is indeed free of all the artifacts exhibited by previous methods (see Figures 14, 15, 16 and 17).

In order to compare computational performance, we have implemented both CPU and GPU versions of dual quaternion skinning.

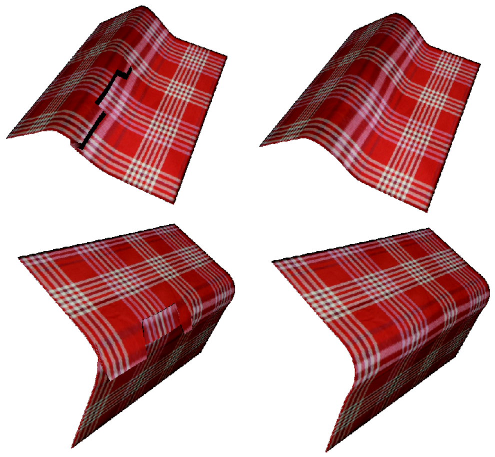


Figure 15: Comparison of direct quaternion blending (left) and dual quaternion (right) blending. Only the latter delivers smooth deformation.

The average performance of the CPU implementations is reported in Figure 18 and the number of instructions of our vertex shaders can be found in Figure 19. Note that our implementation of log-matrix blending uses an optimization for rigid transformations based on the Rodrigues formula, as suggested in [Alexa 2002]. Our vertex shader assumes $n = 4$ (which is common due to graphics hardware considerations) and does not perform any optimizations if there are fewer than 4 influencing joints.

From the measurements, we see that dual quaternion, linear and direct quaternion blending [Hejl 2004] have quite similar performance both on the GPU and CPU. Although our algorithm is slightly slower than both linear and direct quaternion blending, we believe that this is not a high price to pay for the elimination of artifacts. When compared to log-matrix and spherical blending, we see that dual quaternion skinning is more than twice as fast (and also much easier to implement).

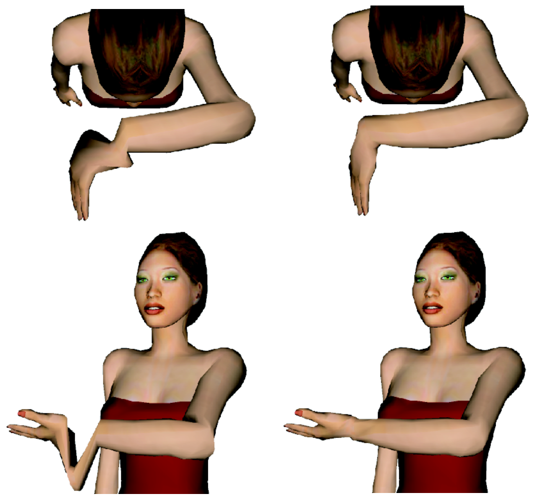


Figure 16: Comparison of log-matrix (left) and dual quaternion (right) blending. The shortest-path property of dual quaternion blending guarantees natural skin deformations.

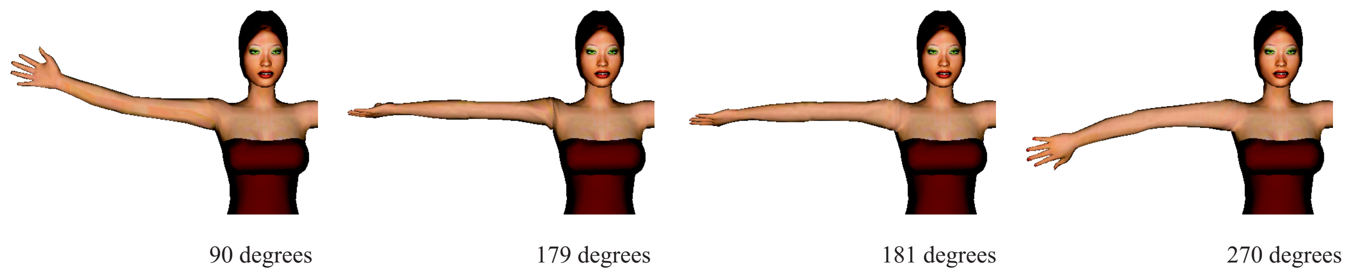


Figure 20: Example of a skin flipping artifact caused by the shortest path property. When rotating from 179 to 181 degrees, skin discontinuously changes its shape, because the other rotation direction becomes shorter.

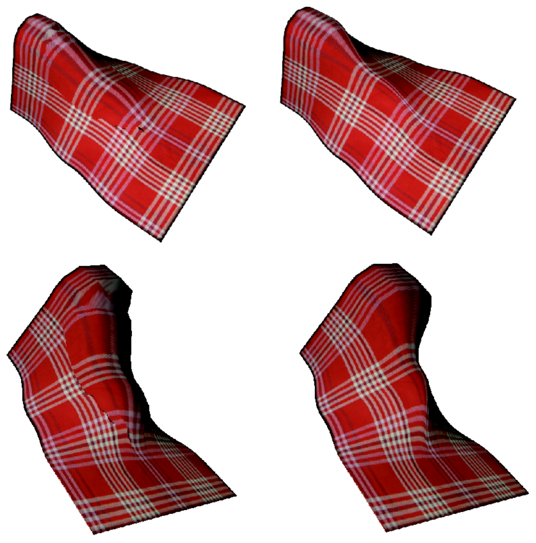


Figure 17: Comparison of spherical (left) and dual quaternion (right) blending. Dual quaternions do not need to cluster vertices and therefore naturally avoid artifacts.

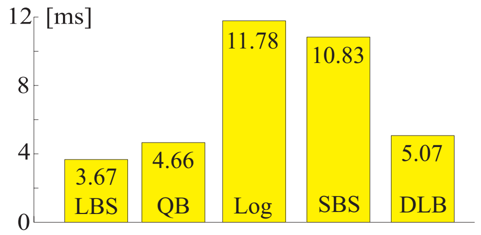


Figure 18: Average CPU performance for skin deformation of the human model in milliseconds (Pentium 4, 3.4 GHz): LBS – linear blend skinning, QB – direct quaternion blending, Log – log-matrix blending, SBS – spherical blend skinning, DLB – dual quaternion linear blending (pre-computed antipodality).

Linear blend skinning supports non-rigid transformations at no extra cost, which is a big advantage. Previous techniques, i.e., direct quaternion blending, log-matrix blending and spherical blend skinning, considered only rigid transformations. Our method to add scale/shear transformations presented in Section 4.2 requires an extra set of 3×4 matrices to be passed to the GPU (along with dual quaternions), which means that we need in total 20 scalars per joint

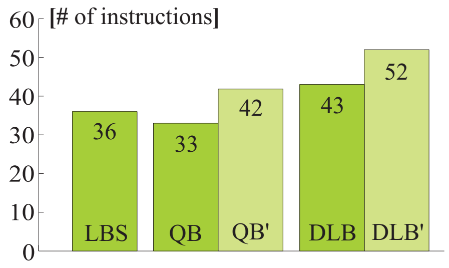


Figure 19: Number of vertex shader instructions: LBS – linear blend skinning, QB – direct quaternion blending (pre-computed antipodality), DLB – dual quaternion linear blending (pre-computed antipodality), QB$'$ and DLB$'$ – same as before but with antipodality resolved in the vertex shader.

(compare with the 12 required by linear blend skinning and the 8 required by standard dual quaternion skinning).

The two-phase skinning also has an impact on the run-time performance, requiring an extra 29 shader instructions to blend matrices $C'_1, \ldots, C'_n$ (for $n = 4$). This means that we end up with about twice as many instructions as required for linear blend skinning. This may or may not be an issue, depending on where the bottlenecks of our application are and what the target hardware is.

The comparison of dual quaternion skinning to the Animation Space method [Merry et al. 2006] is somewhat troublesome. According to our discussion with the author of Animation Space [Merry 2007], it is likely that no objective experimental comparison is possible, because the modelling methods are different (in particular, Animation Space makes use of additional examples). Therefore, it would be hard to judge if a particular result was due to the properties of the algorithm itself or simply how much effort went into the modelling.

6 Conclusions and Future Work

In this paper, we propose a novel skinning method based on the blending of dual quaternions. Our method is efficient, simple to implement and does not require the modification of existing models or authoring tools (advanced methods exploiting the linearity of linear blend skinning, such as [Mohr and Gleicher 2003], could be adapted). We therefore believe that it provides a highly practical alternative to the popular but inaccurate linear blend skinning method.

However, the proposed algorithm has some limitations. When non-rigid joint transformations are required, our method requires longer shader code and more memory than linear blend skinning. Another potential shortcoming of dual quaternion skinning could be a "flipping artifact", which occurs with joint rotations of more than 180 degrees, see Figure 20. This is a corollary of the shortest path property: when the other path becomes shorter, the skin changes its shape discontinuously. An alternative would be to support multiple revolutions and count the number of twists. Even though this would produce continuous deformations, it is doubtful whether the final animation would look more natural. In most situations, however, this is not an issue as extreme rotations leading to flipping artifacts are usually prevented via joint constraints.

Taking a broader perspective, blending of dual quaternions has potentially many more applications, not limited to skinning. More generally, it is a method for rigid transformation blending with interesting properties. Therefore, it could potentially be useful in contexts such as motion blending, analysis or compression [Alexa 2002]. The investigation of other applications of dual quaternions in computer graphics promises to be an interesting area for future work.

6.1 Implementation

In order to facilitate further research and experiments, we have provided some on-line resources:
`http://isg.cs.tcd.ie/projects/DualQuaternions/`
Specifically, C source code converting between dual quaternions and $(quaternion, translation)$ pairs is available, along with Cg vertex shaders that implement our skinning methods.

7 Acknowledgements

We wish to thank the anonymous reviewers for their valuable comments, Bruce Merry for discussion on Animation Space and Carlo H. Séquin for early insights into the topic. We acknowledge the support of the Higher Education Authority of Ireland and Science Foundation Ireland. This work has been partly supported by the Ministry of Education of the Czech Republic under the research programs LC-06008 (Center for Computer Graphics) and MSM 6840770014.

A Dual Quaternion Tutorial

Dual quaternions are not a standard tool in computer graphics, in contrast to regular quaternions. Basic information about them can be found in the literature [Bottema and Roth 1979; McCarthy 1990], which provides a good theoretical background, but without information about applications in computer graphics. In order to bridge this gap, in this appendix we provide a brief tutorial summarizing the properties which are most important to computer graphics practitioners. We assume that the reader is already familiar with regular quaternions, otherwise see for example [Dam et al. 1998; Hanson 2006]. Dual quaternions can be considered as quaternions whose elements are dual numbers. Let us therefore start our discussion with this simpler algebra.

A.1 Dual Numbers

The algebra of dual numbers, denoted as $\hat{R}$, 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

$$(a_0 + \varepsilon a_\varepsilon)(b_0 + \varepsilon b_\varepsilon) = a_0 b_0 + \varepsilon(a_0 b_\varepsilon + a_\varepsilon b_0) \qquad (17)$$

The following lemma establishes that the dual conjugate behaves like the complex conjugate with respect to multiplication.

Lemma 6. *For any $\hat{a}, \hat{b} \in \hat{R}$, it is true that $\overline{\hat{a}\hat{b}} = \overline{\hat{a}}\,\overline{\hat{b}}$.*

Proof.

$$\begin{aligned}\overline{\hat{a}\hat{b}} &= \overline{a_0 b_0 + \varepsilon(a_0 b_\varepsilon + a_\varepsilon b_0)} = \\ &= a_0 b_0 - \varepsilon(a_0 b_\varepsilon + a_\varepsilon b_0) = (a_0 - \varepsilon a_\varepsilon)(b_0 - \varepsilon b_\varepsilon) = \overline{\hat{a}}\,\overline{\hat{b}}\end{aligned}$$

□

The following lemmas present the formulas for dual quaternion inversion and square root.

Lemma 7. *The inverse of a dual number $a_0 + \varepsilon a_\varepsilon \in \hat{R}$, where $a_0 \neq 0$, is given as*

$$\frac{1}{a_0 + \varepsilon a_\varepsilon} = \frac{1}{a_0} - \varepsilon \frac{a_\varepsilon}{a_0^2} \qquad (18)$$

Proof. In order to find the inverse of a dual number $a_0 + \varepsilon a_\varepsilon$, we have to solve for b_0, b_ε in the following equation

$$(a_0 + \varepsilon a_\varepsilon)(b_0 + \varepsilon b_\varepsilon) = 1$$

Using Equation (17), this reduces to the following two equations:

$$\begin{aligned} a_0 b_0 &= 1 \\ a_0 b_\varepsilon + a_\varepsilon b_0 &= 0 \end{aligned}$$

Therefore, $b_0 = 1/a_0$ and $b_\varepsilon = -a_\varepsilon / a_0^2$, provided that $a_0 \neq 0$. □

Note that 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.

Lemma 8. *The square root of a dual number $a_0 + \varepsilon a_\varepsilon \in \hat{R}$, such that $a_0 > 0$ is given as*

$$\sqrt{a_0 + \varepsilon a_\varepsilon} = \sqrt{a_0} + \varepsilon \frac{a_\varepsilon}{2\sqrt{a_0}} \qquad (19)$$

Proof. To find the square root of a dual number $a_0 + \varepsilon a_\varepsilon$, we have to solve the following equation

$$(b_0 + \varepsilon b_\varepsilon)^2 = a_0 + \varepsilon a_\varepsilon$$

Writing out the non-dual and dual components, we obtain

$$b_0^2 = a_0,\ 2b_0 b_\varepsilon = a_\varepsilon$$

from which the formula for dual square root readily follows (provided that $a_0 > 0$). □

Finally, note that the Taylor series of a function of dual argument reduces to a finite sum

$$f(a_0 + \varepsilon a_\varepsilon) = f(a_0) + \varepsilon a_\varepsilon f'(a_0)$$

because the higher powers of ε are zero. For example, dual sine and cosine functions are therefore given as

$$\begin{aligned} \sin(a_0 + \varepsilon a_\varepsilon) &= \sin(a_0) + \varepsilon a_\varepsilon \cos(a_0) \qquad (20) \\ \cos(a_0 + \varepsilon a_\varepsilon) &= \cos(a_0) - \varepsilon a_\varepsilon \sin(a_0) \qquad (21) \end{aligned}$$

A.2 Dual Quaternions

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 $i\varepsilon = \varepsilon i$. Just like ordinary quaternions, dual quaternions are also associative, distributive, but not commutative. A dual quaternion can also be 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: $\hat{\mathbf{q}}^* = \mathbf{q}_0^* + \varepsilon\mathbf{q}_\varepsilon^*$. However, dual number conjugation also applies to dual quaternions: $\overline{\hat{\mathbf{q}}} = \mathbf{q}_0 - \varepsilon\mathbf{q}_\varepsilon$, so we end up with two different conjugations (and we will actually need both of them). It is not difficult to verify that the order in which the conjugations are performed does not matter

$$\overline{\hat{\mathbf{q}}^*} = \overline{\mathbf{q}_0^* + \varepsilon\mathbf{q}_\varepsilon^*} = \mathbf{q}_0^* - \varepsilon\mathbf{q}_\varepsilon^* = (\mathbf{q}_0 - \varepsilon\mathbf{q}_\varepsilon)^* = \overline{\hat{\mathbf{q}}}^*$$

The norm of a dual quaternion is defined as $\|\hat{\mathbf{q}}\| = \sqrt{\hat{\mathbf{q}}^*\hat{\mathbf{q}}} = \sqrt{\hat{\mathbf{q}}\hat{\mathbf{q}}^*}$. It is possible to simplify this expression, as is shown in the following lemma.

Lemma 9. *For any dual quaternion $\hat{\mathbf{q}} = \mathbf{q}_0 + \varepsilon\mathbf{q}_\varepsilon$, such that $\mathbf{q}_0 \neq 0$, the norm can be written as*

$$\|\hat{\mathbf{q}}\| = \|\mathbf{q}_0\| + \varepsilon\frac{\langle\mathbf{q}_0, \mathbf{q}_\varepsilon\rangle}{\|\mathbf{q}_0\|} \tag{22}$$

Proof. Let us first expand

$$\begin{aligned}\hat{\mathbf{q}}^*\hat{\mathbf{q}} &= (\mathbf{q}_0^* + \varepsilon\mathbf{q}_\varepsilon^*)(\mathbf{q}_0 + \varepsilon\mathbf{q}_\varepsilon) = \mathbf{q}_0^*\mathbf{q}_0 + \varepsilon(\mathbf{q}_\varepsilon^*\mathbf{q}_0 + \mathbf{q}_0^*\mathbf{q}_\varepsilon) = \\ &= \|\mathbf{q}_0\|^2 + 2\varepsilon\langle\mathbf{q}_0, \mathbf{q}_\varepsilon\rangle\end{aligned}$$

By taking the dual square root (Equation (19)), we obtain the final formula. □

Unit dual quaternions are those satisfying $\|\hat{\mathbf{q}}\| = 1$. According to the previous lemma, a dual quaternion $\hat{\mathbf{q}}$ is unit if and only if $\|\mathbf{q}_0\| = 1$ and $\langle\mathbf{q}_0, \mathbf{q}_\varepsilon\rangle = 0$. We denote the set of unit dual quaternions as $\hat{\mathcal{Q}}_1$. Geometrically, $\hat{\mathcal{Q}}_1$ is a 6-dimensional manifold embedded in 8-dimensional Euclidean space (called an image-space of dual quaternions [McCarthy 1990]).

Dual quaternions inherit many properties of regular quaternions. The following lemma states that conjugation of dual quaternions also swaps the order of multiplication (as is the case with regular quaternions).

Lemma 10. *For any dual quaternions $\hat{\mathbf{p}}, \hat{\mathbf{q}}$, it is true that*

$$(\hat{\mathbf{p}}\hat{\mathbf{q}})^* = \hat{\mathbf{q}}^*\hat{\mathbf{p}}^* \tag{23}$$

Proof. This is easy to verify by direct computation

$$\begin{aligned}(\hat{\mathbf{p}}\hat{\mathbf{q}})^* &= (\mathbf{p}_0\mathbf{q}_0 + \varepsilon(\mathbf{p}_\varepsilon\mathbf{q}_0 + \mathbf{p}_0\mathbf{q}_\varepsilon))^* = \\ &= (\mathbf{p}_0\mathbf{q}_0)^* + \varepsilon((\mathbf{p}_\varepsilon\mathbf{q}_0)^* + (\mathbf{p}_0\mathbf{q}_\varepsilon)^*) = \\ &= \mathbf{q}_0^*\mathbf{p}_0^* + \varepsilon(\mathbf{q}_0^*\mathbf{p}_\varepsilon^* + \mathbf{q}_\varepsilon^*\mathbf{p}_0^*) = \hat{\mathbf{q}}^*\hat{\mathbf{p}}^*\end{aligned}$$

□

Another important property (again analogous to regular quaternions) is so called multiplicative property of the dual quaternion norm.

Lemma 11. *For any dual quaternions $\hat{\mathbf{p}}, \hat{\mathbf{q}}$, it is true that*

$$\|\hat{\mathbf{p}}\hat{\mathbf{q}}\| = \|\hat{\mathbf{p}}\|\|\hat{\mathbf{q}}\| \tag{24}$$

Proof. Using Lemma 10 and the fact that a dual number commutes with a dual quaternion, we can compute

$$\|\hat{\mathbf{p}}\hat{\mathbf{q}}\|^2 = (\hat{\mathbf{p}}\hat{\mathbf{q}})^*(\hat{\mathbf{p}}\hat{\mathbf{q}}) = \hat{\mathbf{q}}^*\hat{\mathbf{p}}^*\hat{\mathbf{p}}\hat{\mathbf{q}} = \|\hat{\mathbf{p}}\|^2\hat{\mathbf{q}}^*\hat{\mathbf{q}} = \|\hat{\mathbf{p}}\|^2\|\hat{\mathbf{q}}\|^2$$

Taking the dual square root on both sides concludes the proof. □

The inverse of a dual quaternion is defined only when $\mathbf{q}_0 \neq 0$. In this case, we have

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

Note that the inverse of a unit dual quaternion is just conjugation.

Let us now turn our attention to the representation of rigid transformations using dual quaternions. As expected, unit dual quaternions naturally represent 3D rotation, when the dual part $\mathbf{q}_\varepsilon = 0$. 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 $\hat{\mathbf{q}}$ can then be written as $\hat{\mathbf{q}}\hat{\mathbf{v}}\overline{\hat{\mathbf{q}}^*}$. This can be easily verified, because if $\mathbf{q}_\varepsilon = 0$ 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 a regular quaternion.

What is interesting is that dual quaternion multiplication 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, analogous to classical quaternions, which work with half of the angle of rotation). To see this, let us expand

$$\begin{aligned}\hat{\mathbf{t}}\hat{\mathbf{v}}\overline{\hat{\mathbf{t}}^*} &= \hat{\mathbf{t}}(1 + \varepsilon(v_0 i + v_1 j + v_2 k))\left(1 + \frac{\varepsilon}{2}(t_0 i + t_1 j + t_2 k)\right) = \\ &= \hat{\mathbf{t}}\left(1 + \varepsilon\left(\left(v_0 + \frac{t_0}{2}\right)i + \left(v_1 + \frac{t_1}{2}\right)j + \left(v_2 + \frac{t_2}{2}\right)k\right)\right) = \\ &= 1 + \varepsilon((v_0 + t_0)i + (v_1 + t_1)j + (v_2 + t_2)k),\end{aligned}$$

which shows that the unit dual quaternion $\hat{\mathbf{t}}$ performs translation by (t_0, t_1, t_2).

General rigid transformation is a composition of rotation and translation. Therefore, let us first apply the regular quaternion $\mathbf{q}_0$ (representing the rotational component) and then the dual quaternion $\hat{\mathbf{t}}$ (representing the translational component)

$$\hat{\mathbf{t}}(\mathbf{q}_0\hat{\mathbf{v}}\overline{\mathbf{q}_0{}^*})\overline{\hat{\mathbf{t}}^*} = (\hat{\mathbf{t}}\mathbf{q}_0)\hat{\mathbf{v}}(\overline{\mathbf{q}_0{}^*}\,\overline{\hat{\mathbf{t}}^*}) = (\hat{\mathbf{t}}\mathbf{q}_0)\hat{\mathbf{v}}(\overline{\mathbf{q}_0{}^*\hat{\mathbf{t}}^*}) = (\hat{\mathbf{t}}\mathbf{q}_0)\hat{\mathbf{v}}\overline{(\hat{\mathbf{t}}\mathbf{q}_0)^*}$$

Therefore, we can see that $\hat{\mathbf{t}}\mathbf{q}_0$ is a dual quaternion that performs rigid transformation. Expanding, we obtain

$$\hat{\mathbf{t}}\mathbf{q}_0 = \left(1 + \frac{\varepsilon}{2}(t_0 i + t_1 j + t_2 k)\right)\mathbf{q}_0 = \mathbf{q}_0 + \frac{\varepsilon}{2}(t_0 i + t_1 j + t_2 k)\mathbf{q}_0 \tag{25}$$

Now we can state the following:

Lemma 12. *Every rigid transformation can be represented by a unit dual quaternion, and conversely, every unit dual quaternion represents a rigid transformation.*

Proof. The first part of the statement follows from Equation (25), if we can show that $\hat{\mathbf{t}}\mathbf{q}_0$ is unit. However, both $\hat{\mathbf{t}}$ and $\mathbf{q}_0$ are unit and therefore, using Equation (24), $\|\hat{\mathbf{t}}\mathbf{q}_0\| = \|\hat{\mathbf{t}}\|\|\mathbf{q}_0\| = 1$.

To prove the second part, we consider a unit dual quaternion $\hat{\mathbf{p}} = \mathbf{p}_0 + \varepsilon \mathbf{p}_\varepsilon$. We need to find (t_0, t_1, t_2) and $\mathbf{q}_0$ so that

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

Obviously, $\mathbf{q}_0 = \mathbf{p}_0$ and (t_0, t_1, t_2) is given by the equation $\frac{1}{2}(t_0 i + t_1 j + t_2 k)\mathbf{p}_0 = \mathbf{p}_\varepsilon$. This equation can be solved if we show that the scalar part of $\mathbf{p}_\varepsilon \mathbf{p}_0^*$ is zero. However, the scalar part of $\mathbf{p}_\varepsilon \mathbf{p}_0^*$ is equivalent to $\langle \mathbf{p}_\varepsilon, \mathbf{p}_0 \rangle$, which is zero because $\hat{\mathbf{p}}$ is unit. □

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. Equation (25) 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 by the vector part of $2\mathbf{q}_\varepsilon \mathbf{q}_0^*$. Note that the conversion routines in the C language are available on our website (see Section 6.1).

Dual quaternion conjugations can be interpreted as follows. Unit dual quaternion conjugate $\hat{\mathbf{q}}^*$ corresponds to the inverse transformation of $\hat{\mathbf{q}}$, as can be seen from Equation (25). The same equation reveals that dual-number-like conjugation, $\overline{\hat{\mathbf{q}}}$, inverts translation only, leaving rotation intact. Both conjugations applied together, $\overline{\hat{\mathbf{q}}}^*$, correspond to the inverse rigid transformation but with the original translational part.

A.3 Connection to Spatial Kinematics

The geometric interpretation of regular quaternions follows from the formula $\mathbf{q} = \cos\frac{\theta}{2} + \mathbf{s}\sin\frac{\theta}{2}$, which contains the axis of rotation $\mathbf{s}$ and the angle of rotation θ. The following lemma generalizes that to dual quaternions. In spite of the lengthier proof, it is important because it reveals the geometrical interpretation of dual quaternions and their connection to spatial kinematics (Chasles' theorem).

Lemma 13. *Let $\hat{\theta} \in \hat{R}$ and $\hat{\mathbf{s}} \in \hat{Q}_1$ with zero scalar part. Then*

$$\hat{\mathbf{q}} = \cos\frac{\hat{\theta}}{2} + \hat{\mathbf{s}}\sin\frac{\hat{\theta}}{2} \tag{26}$$

is a unit dual quaternion. Conversely, for every $\hat{\mathbf{q}} \in \hat{Q}_1$, there exists $\hat{\theta} \in \hat{R}$ and $\hat{\mathbf{s}} \in \hat{Q}_1$ with zero scalar part such that Equation (26) holds.

Proof. First, let us show that $\hat{\mathbf{q}}$ is always unit. Therefore, we expand Equation (26) to give

$$\hat{\mathbf{q}} = \cos\frac{\theta_0 + \varepsilon\theta_\varepsilon}{2} + (\mathbf{s}_0 + \varepsilon \mathbf{s}_\varepsilon)\sin\frac{\theta_0 + \varepsilon\theta_\varepsilon}{2}$$

and using Equations (20) and (21)

$$\hat{\mathbf{q}} = \cos\frac{\theta_0}{2} - \varepsilon\frac{\theta_\varepsilon}{2}\sin\frac{\theta_0}{2} + (\mathbf{s}_0 + \varepsilon\mathbf{s}_\varepsilon)\left(\sin\frac{\theta_0}{2} + \varepsilon\frac{\theta_\varepsilon}{2}\cos\frac{\theta_0}{2}\right)$$

isolating the non-dual and dual parts

$$\hat{\mathbf{q}} = \underbrace{\cos\frac{\theta_0}{2} + \mathbf{s}_0\sin\frac{\theta_0}{2}}_{\mathbf{q}_0} + \varepsilon\underbrace{\left(\mathbf{s}_0\frac{\theta_\varepsilon}{2}\cos\frac{\theta_0}{2} - \frac{\theta_\varepsilon}{2}\sin\frac{\theta_0}{2} + \mathbf{s}_\varepsilon\sin\frac{\theta_0}{2}\right)}_{\mathbf{q}_\varepsilon}$$

To show that $\hat{\mathbf{q}} = \mathbf{q}_0 + \varepsilon\mathbf{q}_\varepsilon$ is unit, we have to show that $\|\mathbf{q}_0\| = 1$ and $\langle \mathbf{q}_0, \mathbf{q}_\varepsilon\rangle = 0$. Quaternion $\mathbf{q}_0$ is obviously unit (because $\mathbf{s}_0$ is). To verify the second condition, we compute

$$\begin{aligned}\langle \mathbf{q}_0, \mathbf{q}_\varepsilon\rangle &= \left\langle \mathbf{s}_0\sin\frac{\theta_0}{2}, \mathbf{s}_0\frac{\theta_\varepsilon}{2}\cos\frac{\theta_0}{2} + \mathbf{s}_\varepsilon\sin\frac{\theta_0}{2}\right\rangle - \frac{\theta_\varepsilon}{2}\sin\frac{\theta_0}{2}\cos\frac{\theta_0}{2} \\ &= \frac{\theta_\varepsilon}{2}\sin\frac{\theta_0}{2}\cos\frac{\theta_0}{2} - \frac{\theta_\varepsilon}{2}\sin\frac{\theta_0}{2}\cos\frac{\theta_0}{2} = 0\end{aligned}$$

because $\langle \mathbf{s}_0, \mathbf{s}_0\rangle = 1$ and $\langle \mathbf{s}_0, \mathbf{s}_\varepsilon\rangle = 0$.

To prove the second part of the statement, let us assume that $\hat{\mathbf{q}} \in \hat{Q}_1$ is fixed. The task is to find $\hat{\theta} = \theta_0 + \varepsilon\theta_\varepsilon$ and $\hat{\mathbf{s}} = \mathbf{s}_0 + \varepsilon\mathbf{s}_\varepsilon$ so that Equation (26) holds. According to Lemma 12, we know that $\hat{\mathbf{q}}$ can be written as $\hat{\mathbf{q}} = \mathbf{q}_0 + \frac{\varepsilon}{2}\mathbf{t}\mathbf{q}_0$. As $\mathbf{q}_0 = \cos\frac{\theta_0}{2} + \mathbf{s}_0\sin\frac{\theta_0}{2}$, the unknowns θ_0 and $\mathbf{s}_0$ are given by converting $\mathbf{q}_0$ to an axis-angle representation. Now we must find θ_ε and $\mathbf{s}_\varepsilon$, which are given by

$$\frac{1}{2}\mathbf{t}\mathbf{q}_0 = \mathbf{s}_0\frac{\theta_\varepsilon}{2}\cos\frac{\theta_0}{2} - \frac{\theta_\varepsilon}{2}\sin\frac{\theta_0}{2} + \mathbf{s}_\varepsilon\sin\frac{\theta_0}{2} \tag{27}$$

Let us expand the left-hand side, i.e., write out the quaternion product using Equation (14)

$$\begin{aligned}\frac{1}{2}\mathbf{t}\mathbf{q}_0 &= \frac{1}{2}\mathbf{t}\left(\cos\frac{\theta_0}{2} + \mathbf{s}_0\sin\frac{\theta_0}{2}\right) = \\ &= \frac{1}{2}\mathbf{t}\cos\frac{\theta_0}{2} - \frac{1}{2}\sin\frac{\theta_0}{2}\langle \mathbf{t}, \mathbf{s}_0\rangle + \frac{1}{2}(\mathbf{t}\times\mathbf{s}_0)\sin\frac{\theta_0}{2}\end{aligned} \tag{28}$$

We will consider three cases: (i) $\mathbf{q}_0 = 1$, (ii) $\mathbf{q}_0 = -1$, (iii) $\mathbf{q}_0 \neq \pm 1$. In the first case, we can set $\theta_0 = 0$. The Equations (27) and (28) thus reduce to

$$\frac{\theta_\varepsilon}{2}\mathbf{s}_0 = \frac{1}{2}\mathbf{t}$$

Therefore, it is sufficient to set $\theta_\varepsilon = \|\mathbf{t}\|$ and $\mathbf{s}_0 = \mathbf{t}/\|\mathbf{t}\|$, while $\mathbf{s}_\varepsilon$ can be, for example, the zero vector. The second case is quite similar, except that we assume $\theta_0 = 2\pi$. The choice of $\theta_\varepsilon, \mathbf{s}_0$ and $\mathbf{s}_\varepsilon$ is the same as in the previous case.

In the last case, when $\mathbf{q}_0 \neq \pm 1$, we know that $\theta_0 \neq 2k\pi$ for any integer k. By comparing the scalar parts of Equations (27) and (28), we obtain

$$-\frac{\theta_\varepsilon}{2}\sin\frac{\theta_0}{2} = -\frac{1}{2}\sin\frac{\theta_0}{2}\langle \mathbf{t}, \mathbf{s}_0\rangle$$

Since $\sin\frac{\theta_0}{2} \neq 0$, this simplifies to $\theta_\varepsilon = \langle \mathbf{t}, \mathbf{s}_0\rangle$, yielding the formula for θ_ε. Comparison of the vector parts of Equations (27) and (28) gives

$$\mathbf{s}_0\frac{\theta_\varepsilon}{2}\cos\frac{\theta_0}{2} + \mathbf{s}_\varepsilon\sin\frac{\theta_0}{2} = \frac{1}{2}\mathbf{t}\cos\frac{\theta_0}{2} + \frac{1}{2}(\mathbf{t}\times\mathbf{s}_0)\sin\frac{\theta_0}{2}$$

This allows us to express $\mathbf{s}_\varepsilon$ explicitly

$$\begin{aligned}\mathbf{s}_\varepsilon\sin\frac{\theta_0}{2} &= \frac{1}{2}(\mathbf{t} - \mathbf{s}_0\theta_\varepsilon)\cos\frac{\theta_0}{2} + \frac{1}{2}(\mathbf{t}\times\mathbf{s}_0)\sin\frac{\theta_0}{2} \\ \mathbf{s}_\varepsilon &= \frac{1}{2}(\mathbf{t} - \mathbf{s}_0\theta_\varepsilon)\operatorname{cotan}\frac{\theta_0}{2} + \frac{1}{2}(\mathbf{t}\times\mathbf{s}_0)\end{aligned}$$

The expression of $\mathbf{s}_\varepsilon$ can be simplified, if we recall that $\theta_\varepsilon = \langle \mathbf{t}, \mathbf{s}_0\rangle$ and, using Lagrange's equation, $\mathbf{t} - \mathbf{s}_0\langle \mathbf{t}, \mathbf{s}_0\rangle = (\mathbf{s}_0\times\mathbf{t})\times\mathbf{s}_0$. This enables us to write

$$\mathbf{s}_\varepsilon = \frac{1}{2}\left((\mathbf{s}_0\times\mathbf{t})\operatorname{cotan}\frac{\theta_0}{2} + \mathbf{t}\right)\times\mathbf{s}_0 \tag{29}$$

From this expression we see that obviously $\langle \mathbf{s}_0, \mathbf{s}_\varepsilon\rangle = 0$ and therefore, $\hat{\mathbf{s}}_0 \in \hat{Q}_1$, as required. □

Intuitively speaking, Lemma 13 states that any unit dual quaternion is composed of parameters $\theta_0, \theta_\varepsilon, \mathbf{s}_0$ and $\mathbf{s}_\varepsilon$. As we can see from the proof, θ_0 is the angle of rotation, and unit vector $\mathbf{s}_0$ represents the direction of the axis of rotation (in the degenerate case with $\theta_0 = 0$ or $\theta_0 = 2\pi$, $\mathbf{s}_0$ represents the direction of the translation vector). Furthermore, $\theta_\varepsilon = \langle \mathbf{t}, \mathbf{s}_0 \rangle$ is the amount of translation along vector $\mathbf{s}_0$. The only slightly less intuitive variable is $\mathbf{s}_\varepsilon$. However, if we recall Equation (4), we observe that the term

$$\mathbf{r} = \frac{1}{2}\left((\mathbf{s}_0 \times \mathbf{t}) \operatorname{cotan} \frac{\theta_0}{2} + \mathbf{t} \right)$$

that occurs in Equation (29) in the form $\mathbf{s}_\varepsilon = \mathbf{r} \times \mathbf{s}_0$ is the center of rotation. Therefore, we can conclude that the variables $\theta_0, \theta_\varepsilon, \mathbf{s}_0$ and $\mathbf{s}_\varepsilon$ are parameters of the associated screw motion.

There is a close connection with a classical result of spatial kinematics known as Chasles' theorem [Murray et al. 1994]. Chasle's theorem states that any rigid transformation can be described by a screw, i.e., rotation about an axis followed by translation in the direction of that axis. For example, in Figure 21 top, we have a teapot which is first rotated about axis $\mathbf{s}_0, \|\mathbf{s}_0\| = 1$, and then translated by vector $\mathbf{t}$. The translation vector $\mathbf{t}$ can be decomposed to $\mathbf{t}_0 = \mathbf{t}_\parallel + \mathbf{t}_\perp$, where $\mathbf{t}_\parallel$ is the component parallel to $\mathbf{s}_0$ and $\mathbf{t}_\perp$ is the component orthogonal to $\mathbf{s}_0$ (formally, $\mathbf{t}_\parallel = \mathbf{s}_0 \langle \mathbf{s}_0, \mathbf{t} \rangle$, $\mathbf{t}_\perp = \mathbf{t} - \mathbf{t}_\parallel$). The component $\mathbf{t}_\perp$ lies in the plane with normal $\mathbf{s}_0$, and therefore can be recovered by selecting the proper center of rotation $\mathbf{r}$ in that plane. If we shift the axis of rotation to point $\mathbf{r}$, as shown in Figure 21 bottom, we obtain the corresponding screw motion (i.e., with translational component reduced to $\mathbf{t}_\parallel$).

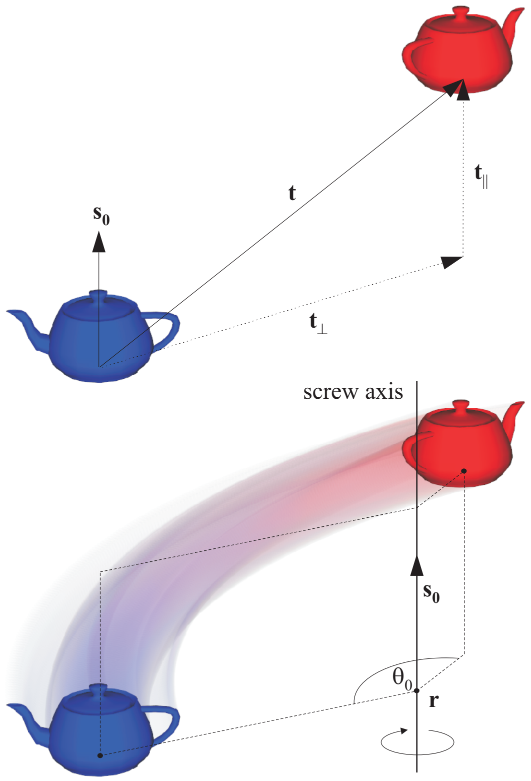


Figure 21: Conversion of a rotation about axis $\mathbf{s}_0, \|\mathbf{s}_0\| = 1$, followed by translation $\mathbf{t}$ (top) to the corresponding screw (bottom).

We can conclude that dual quaternions are a special representation of the screw parameters – one with advantageous algebraical properties.

A.4 Exponential and Logarithm

Equation (26) can be used to define the exponential of any dual quaternion $\hat{\mathbf{p}}$ with zero scalar part and non-zero non-dual part. Let us introduce $\hat{\mathbf{s}} = \hat{\mathbf{p}}/\|\hat{\mathbf{p}}\|$ and $\hat{\theta} = 2\|\hat{\mathbf{p}}\|$, so that $\hat{\mathbf{p}} = \hat{\mathbf{s}}\frac{\hat{\theta}}{2}$ and $\hat{\mathbf{s}}$ is a unit dual quaternion with zero scalar part. Then, the exponential can be defined as follows

$$\exp\, \hat{\mathbf{p}} = \exp\left(\hat{\mathbf{s}}\frac{\hat{\theta}}{2}\right) = \cos\frac{\hat{\theta}}{2} + \hat{\mathbf{s}} \sin\frac{\hat{\theta}}{2}$$

As shown in Lemma 13, exp $\hat{\mathbf{p}}$ is a unit dual quaternion. The inverse mapping, from unit dual quaternions to dual quaternions with zero scalar part, is denoted as log. The following lemma establishes that dual quaternion exponential and logarithm behave in a similar way to their matrix counterparts.

Lemma 14. *Let $\hat{\mathbf{q}} = \cos\frac{\hat{\theta}}{2} + \hat{\mathbf{s}}\sin\frac{\hat{\theta}}{2}$, where $\hat{\mathbf{q}}, \hat{\mathbf{s}} \in \hat{Q}_1$ and $\hat{\mathbf{s}}$ has zero scalar part. Then for any $\hat{\mathbf{m}} \in \hat{Q}_1$, both of the following equations are true*

$$\exp\left(\hat{\mathbf{m}}\hat{\mathbf{s}}\frac{\hat{\theta}}{2}\hat{\mathbf{m}}^*\right) = \hat{\mathbf{m}} \exp\left(\hat{\mathbf{s}}\frac{\hat{\theta}}{2}\right)\hat{\mathbf{m}}^* \qquad (30)$$

$$\log(\hat{\mathbf{m}}\hat{\mathbf{q}}\hat{\mathbf{m}}^*) = \hat{\mathbf{m}} \log(\hat{\mathbf{q}})\hat{\mathbf{m}}^* \qquad (31)$$

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. This means that the exp on the left hand side of the first equation is well defined and, according to its definition, we can write

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

because a dual number always commutes with a dual quaternion and $\hat{\mathbf{m}}\hat{\mathbf{m}}^* = 1$. This proves Equation (30). The proof of Equation (31) is similar

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

Therefore $\log(\hat{\mathbf{m}}\hat{\mathbf{q}}\hat{\mathbf{m}}^*) = \hat{\mathbf{m}}\hat{\mathbf{s}}\frac{\hat{\theta}}{2}\hat{\mathbf{m}}^* = \hat{\mathbf{m}}\log(\hat{\mathbf{q}})\hat{\mathbf{m}}^*$. $\square$

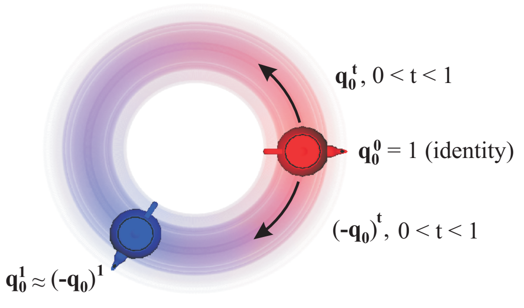
A power of a unit dual quaternion $\hat{\mathbf{q}}$ is then defined naturally:

$$\hat{\mathbf{q}}^{\hat{u}} = \exp(\hat{u}\log\hat{\mathbf{q}}) = \cos\left(\hat{u}\frac{\hat{\theta}}{2}\right) + \hat{\mathbf{s}}\sin\left(\hat{u}\frac{\hat{\theta}}{2}\right)$$

Dual quaternions exhibit so-called *antipodality*, i.e., the fact that both $\hat{\mathbf{q}}$ and $-\hat{\mathbf{q}}$ represent the same rigid transformation

$$(-\hat{\mathbf{q}})\hat{\mathbf{v}}\overline{(-\hat{\mathbf{q}})^*} = (-\hat{\mathbf{q}})\hat{\mathbf{v}}(-\overline{\hat{\mathbf{q}}^*}) = \hat{\mathbf{q}}\hat{\mathbf{v}}\overline{\hat{\mathbf{q}}^*}$$

The mapping between $SE(3)$ and $\hat{Q}_1$ is thus one to two. Note that this is equivalent to the antipodality of regular quaternions. However, 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 other to counterclockwise rotation, see Figure 22. Therefore, when converting matrices to dual quaternions, we must choose an appropriate sign. This depends on each particular application; our method to resolve antipodality in skinning is discussed in Section 4.1.



$\mathbf{q}_0^t$, 0 < t < 1
$\mathbf{q}_0^0$ = 1 (identity)
$(-\mathbf{q}_0)^t$, 0 < t < 1
$\mathbf{q}_0^1 \approx (-\mathbf{q}_0)^1$

Figure 22: 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).

B Difference between DLB and ScLERP

As stated in Section 3.4, the problem of comparing DLB and ScLERP requires derivation of the dual angle $\hat{\beta}_t$ from Equation (13) and then comparing it with $\hat{\alpha}t$ for $t \in [0,1]$. For brevity, we will use shorthand C for $\cos(\frac{\alpha_0}{2})$ and S for $\sin(\frac{\alpha_0}{2})$. Using this convention, Equations (20) and (21) take the following form

$$\cos\left(\frac{\hat{\alpha}}{2}\right) = C - \varepsilon\frac{\alpha_\varepsilon}{2}S, \quad \sin\left(\frac{\hat{\alpha}}{2}\right) = S + \varepsilon\frac{\alpha_\varepsilon}{2}C$$

Now, we can expand the term in the denominator of Equation (13)

$$\begin{aligned} 1-t+t\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^* &= 1-t+t\cos\left(\frac{\hat{\alpha}}{2}\right)+\hat{\mathbf{n}}t\sin\left(\frac{\hat{\alpha}}{2}\right) = \\ &= 1-t+tC-\varepsilon t\frac{\alpha_\varepsilon}{2}S+(\mathbf{n}_0+\varepsilon\mathbf{n}_\varepsilon)t\left(S+\varepsilon\frac{\alpha_\varepsilon}{2}C\right) = \\ &= \underbrace{1-t+tC+\mathbf{n}_0tS}_{\mathbf{r}_0}+\varepsilon\underbrace{t\left(-\frac{\alpha_\varepsilon}{2}S+\mathbf{n}_\varepsilon S+\frac{\alpha_\varepsilon}{2}\mathbf{n}_0C\right)}_{\mathbf{r}_\varepsilon} \end{aligned}$$

The newly introduced quaternions $\mathbf{r}_0$ and $\mathbf{r}_\varepsilon$ satisfy

$$\begin{aligned} \|\mathbf{r}_0\| &= \sqrt{(1-t+tC)^2+t^2S^2} \\ \langle\mathbf{r}_0,\mathbf{r}_\varepsilon\rangle &= \left\langle 1-t+tC+\mathbf{n}_0tS, -t\frac{\alpha_\varepsilon}{2}S+t\mathbf{n}_\varepsilon S+t\frac{\alpha_\varepsilon}{2}\mathbf{n}_0C\right\rangle \\ &= (t-1-tC)t\frac{\alpha_\varepsilon}{2}S+\frac{\alpha_\varepsilon}{2}t^2CS = (t-1)t\frac{\alpha_\varepsilon}{2}S \end{aligned}$$

because $\hat{\mathbf{n}} = \mathbf{n}_0 + \varepsilon\mathbf{n}_\varepsilon$ is a unit dual quaternion (with zero scalar part). Therefore, the denominator of our equation can be written as

$$\|1-t+t\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\| = \|\mathbf{r}_0+\varepsilon\mathbf{r}_\varepsilon\| = \|\mathbf{r}_0\|+\varepsilon\frac{\langle\mathbf{r}_0,\mathbf{r}_\varepsilon\rangle}{\|\mathbf{r}_0\|}$$

and its inverse

$$\frac{1}{\|1-t+t\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\|} = \frac{1}{\|\mathbf{r}_0\|} - \varepsilon\frac{\langle\mathbf{r}_0,\mathbf{r}_\varepsilon\rangle}{\|\mathbf{r}_0\|^3}$$

which gives

$$\begin{aligned} \cos\frac{\hat{\beta}_t}{2} &= \frac{1-t+t\cos(\frac{\hat{\alpha}}{2})}{\|1-t+t\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\|} = \frac{1-t+tC-\varepsilon t\frac{\alpha_\varepsilon}{2}S}{\|1-t+t\hat{\mathbf{q}}_2\hat{\mathbf{q}}_1^*\|} = \\ &= \frac{1-t+tC}{\|\mathbf{r}_0\|} - \varepsilon\left(\frac{t\alpha_\varepsilon S}{2\|\mathbf{r}_0\|}+\frac{(1-t+tC)\langle\mathbf{r}_0,\mathbf{r}_\varepsilon\rangle}{\|\mathbf{r}_0\|^3}\right) \end{aligned}$$

We denote the function above as $f(t)$. Now, we employ Maple [Char et al. 1983] in order to compute $\hat{\beta}_t$ by taking the arc cosine of $f(t)$. In the listing (see Figure 23), the norm $\|\mathbf{r}_0\|$ is denoted as `r0`, and the non-dual and dual parts of $f(t)$ as `f0` and `fe`. The non-dual component of $\hat{\beta}_t$ is called `ang` and the dual one `pitch`, emphasizing their geometric interpretation. In the first part of the listing, we actually re-compute the result from [Kavan and Žára 2005], showing that the upper bound of the angular difference between QLB and SLERP is 0.143 radians, i.e., 8.15 degrees. In the second part (the dual quaternion-specific one), we derive the difference between the translational parts of DLB and ScLERP, which turns out to be a linear function of α_ε (the input translation). Specifically, the difference between translation of DLB and ScLERP is shown to be always strictly less than $0.151\alpha_\varepsilon$.

B.1 Error Analysis for $n > 2$

As mentioned in Section 3.4, the situation with more than two rigid transformations is considerably more complex. In particular, it is necessary to generalize Buss and Fillmore's spherical averages [2001] to the space of unit dual quaternions (which is not a hypersphere). Such an algorithm has been proposed in [Kavan et al. 2006] and called Dual quaternion Iterative Blending (DIB) – see Algorithm 3.

Algorithm 3 (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 $\hat{\mathbf{b}}$

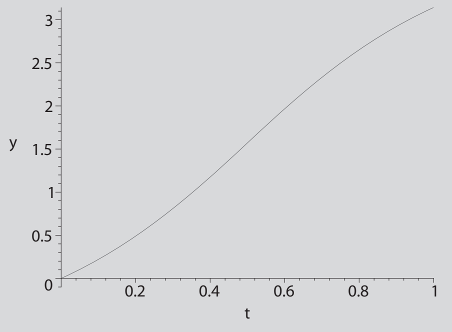
```
$\hat{\mathbf{b}} = DLB(\mathbf{w};\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_n)$
repeat
    $\hat{\mathbf{x}} = \sum_{i=1}^n w_i \log(\hat{\mathbf{b}}^*\hat{\mathbf{q}}_i)$
    $\hat{\mathbf{b}} = \hat{\mathbf{b}}\exp(\hat{\mathbf{x}})$
until $\|\hat{\mathbf{x}}\| < p$
return $\hat{\mathbf{b}}$
```

An intuitive explanation of this algorithm is shown in Figure 24. In the first step, 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 $\hat{Q}_1$ 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.

It is easy to see that DIB is bi-invariant, which follows from the bi-invariance of DLB (Lemma 2) and the properties of dual quaternion exponential and logarithm (Lemma 14). Note that DIB is indeed a generalization of ScLERP. In fact, it can be shown that for $n = 2$, DIB terminates in just a single iteration [Kavan et al. 2006]. For $n > 2$, more iterations are generally required, but in practice, the algorithm converges quite quickly. However, a mathematical analysis of DIB (along the lines of [Buss and Fillmore 2001]) is a non-trivial task that has not been addressed in the literature so far.

It is unfortunately not possible to repeat the error analysis from the previous section for DIB, because we do not have any closed-form expression of DIB. It might still be possible to establish a (potentially non-tight) upper bound. However, that would require a deeper analysis of the geometry of $\hat{Q}_1$ (the case of $n = 2$ corresponds only to a simple subspace of $\hat{Q}_1$, i.e., a screw motion). Note that even an empirical comparison of DLB and DIB (i.e., by sampling all possible rigid transformations) is very challenging for $n > 2$, because of the phenomenon known as the *curse of dimensionality* (i.e., the

```
>  r0 := sqrt((1-t+t*cos(alpha_0/2))^2 +
t*t*sin(alpha_0/2)^2 ):
>  f0 := (t*cos(alpha_0/2) + 1-t)/r0:
>  fe := - t*alpha_e/2*sin(alpha_0/2)/r0 -
(1-t + t*cos(alpha_0/2))*(t-1)*t*alpha_e/2*
sin(alpha_0/2)/r0^3:
>  ang := 2*arccos(f0):
>  plot( subs(alpha_0 = Pi, t -> ang(t)), t =
0..1, y = -0.1..3.14);
```



```
>  anglediff := ang - alpha_0*t:
>  evalf(minimize(subs(alpha_0 = Pi, t ->
anglediff(t)), t = 0..1));
```

$$-.1422292715$$

```
>  evalf(maximize(subs(alpha_0 = Pi, t ->
anglediff(t)), t = 0..1));
```

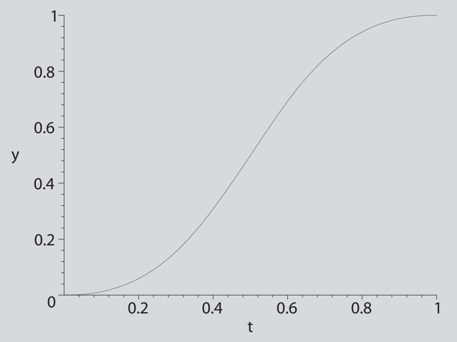
$$.1422292755$$

```
>  pitch := simplify(-2*fe/sin(ang/2));
```

$$pitch := t^2\, alpha\_e \sin(\frac{1}{2}\, alpha\_0)\,(-\%1 - t + t\,\%1) \Big/ ((-1 + 2t - 2t\,\%1 - 2t^2 + 2t^2\,\%1)\sqrt{1 - 2t + 2t\,\%1 + 2t^2 - 2t^2\,\%1}\sqrt{\frac{t^2\,(-1 + \%1^2)}{-1 + 2t - 2t\,\%1 - 2t^2 + 2t^2\,\%1}})$$

$$\%1 := \cos(\frac{1}{2}\, alpha\_0)$$

```
>  plot( subs(alpha_0 = Pi, alpha_e=1, t ->
pitch(t)), t = 0..1, y = -1..1);
```



```
>  pitchdiff := pitch - alpha_e*t:
>  evalf(minimize(subs(alpha_0 = Pi, alpha_e=1,
t -> pitchdiff(t)), t = 0..1));
```

$$-.1501415529$$

```
>  evalf(maximize(subs(alpha_0 = Pi, alpha_e=1,
t -> pitchdiff(t)), t = 0..1));
```

$$.1501415529$$

Figure 23: Upper bound of the difference between *ScLERP* and *DLB* for $n = 2$, computed using Maple.

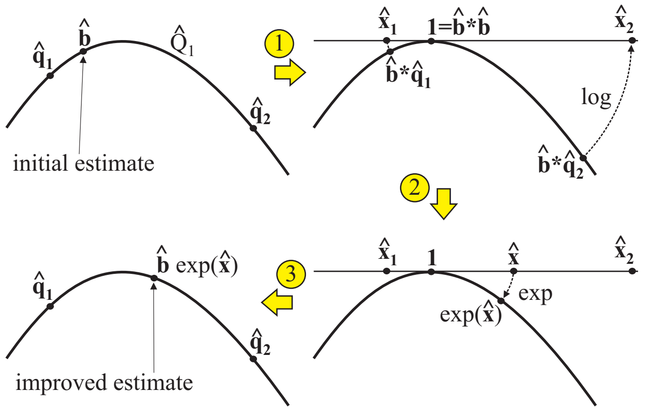


Figure 24: Illustration of one iteration of the DIB algorithm.

fact that the number of required samples grows exponentially with dimension).

In order to determine the error practically, we therefore compare only the results of applying DLB and DIB in skinning. In particular, we compute the maximum difference of vertex positions obtained by DLB and DIB. As expected, we found that the deviation between DLB and DIB is maximal in situations with large joint rotations, such as in Figure 25. However, even in that case, the differences between DLB (Figure 25a) and DIB (Figure 25b) are not noticeable. Therefore, we have visualized the error by color in Figure 25(c, d). Note that in none of our test animations did the error exceed 0.29 units (considering the proportions of our character, the units correspond approximately to inches).

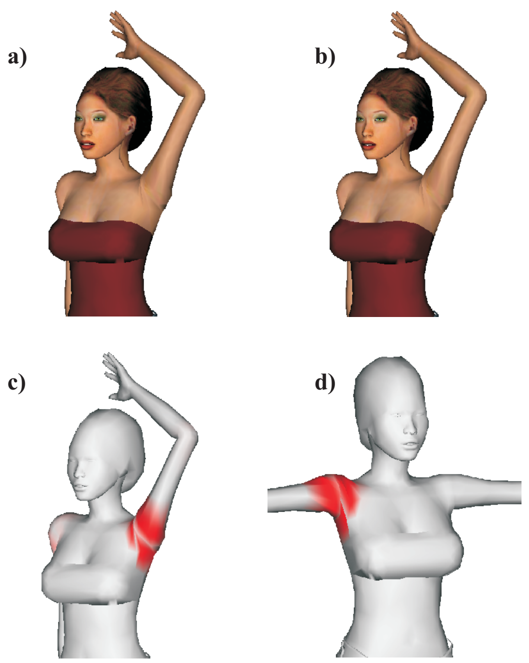


Figure 25: Comparison of DLB and DIB applied in skinning. The difference between DLB (a) and DIB (b) is imperceptible even for large joint rotations and therefore we visualize it using color in (c) and (d).

On the other hand, the difference between computation times of DLB and DIB is quite pronounced. Our CPU implementation takes 5.07ms with DLB and 15.33ms with DIB (with accuracy $p = 10^{-5}$). Therefore, we conclude that the DIB algorithm, albeit theoretically perfect, is not very practical in skinning – the significantly increased cost does not yield significant improvement in deformation quality.

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 POPOVIĆ, 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.

ALLEN, B., CURLESS, B., POPOVIĆ, Z., AND HERTZMANN, A. 2006. Learning a correlated model of identity and pose-dependent body shape variation for real-time synthesis. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, 147–156.

ANGUELOV, D., SRINIVASAN, P., KOLLER, D., THRUN, S., RODGERS, J., AND DAVIS, J. 2005. SCAPE: shape completion and animation of people. *ACM Trans. Graph. 24*, 3, 408–416.

AUBEL, A., AND THALMANN, D. 2000. Realistic deformation of human body shapes. *Proc. Computer Animation and Simulation 2000*, 125–135.

BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3D characters. *ACM Trans. Graph. 26*, 3, 72.

BARR, A. H., CURRIN, B., GABRIEL, S., AND HUGHES, J. F. 1992. Smooth interpolation of orientations with angular velocity constraints using quaternions. *ACM Trans. Graph.*, 313–320.

BELTA, C., AND KUMAR, V. 2002. An SVD-based projection method for interpolation on SE(3). *IEEE Transactions on Robotics and Automation 18*, 3, 334–345.

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.

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

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIC, Z. 2002. Interactive skeleton-driven dynamic deformations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 586–593.

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.

CORDIER, F., AND MAGNENAT-THALMANN, N. 2005. A data-driven approach for real-time clothes simulation. *Computer Graphics Forum 24*, 2, 173–183.

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.

FONTIJNE, D., AND DORST, L. 2003. Modeling 3D euclidean geometry. *IEEE Comput. Graph. Appl. 23*, 2, 68–78.

FORSTMANN, S., AND OHYA, J. 2006. Fast skeletal animation by skinned arc-spline based deformation. In *EG 2006 Short Papers*, 1–4.

FORSTMANN, S., OHYA, J., KROHN-GRIMBERGHE, A., AND MCDOUGALL, R. 2007. Deformation styles for spline-based skeletal animation. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, 141–150.

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

GUO, Z., AND WONG, K. C. 2005. Skinning with deformable chunks. *Computer Graphics Forum 24*, 3, 373–381.

HANSON, A. J. 2006. *Visualizing Quaternions*. Morgan Kaufmann Publishers Inc.

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

HOFER, M., AND POTTMANN, H. 2004. Energy-minimizing splines in manifolds. *ACM Trans. Graph. 23*, 3, 284–293.

HYUN, D.-E., YOON, S.-H., CHANG, J.-W., SEONG, J.-K., KIM, M.-S., AND JÜTTLER, B. 2005. Sweep-based human deformation. *The Visual Computer 21*, 8-10, 542–550.

JACKA, D., REID, A., MERRY, B., AND GAIN, J. 2007. A comparison of linear skinning techniques for character animation. In *AFRIGRAPH '07: Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, ACM, New York, NY, USA, 177–186.

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.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, 71.

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, ACM, New York, NY, USA, 561–566.

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

KAVAN, L., AND ŽÁRA, 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.

KAVAN, L., COLLINS, S., O'SULLIVAN, C., AND ŽÁRA, J., 2006. Dual quaternions for rigid transformation blending. Technical report TCD-CS-2006-46, Trinity College Dublin.

KAVAN, L., COLLINS, S., ŽÁRA, J., AND O'SULLIVAN, C. 2007. Skinning with dual quaternions. In *2007 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press, 39–46.

KRY, P. G., JAMES, D. L., AND PAI, D. K. 2002. Eigenskin: real time large deformation character skinning in hardware. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, 153–159.

KURIHARA, T., AND MIYATA, N. 2004. Modeling deformable human hands from medical images. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 355–363.

KURIHARA, T., AND NISHITA, T. 2007. Dual-quaternion skinning with non-rigid transformatio. In *SCA '07: Posters*, Eurographics Association, Aire-la-Ville, Switzerland, 18–19.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 165–172.

LI, J., AND HAO, P. 2006. Smooth interpolation on homogeneous matrix groups for computer animation. *Journal of Zhejiang University 7*, 7, 1168–1177.

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., LAPERRIÈRE, R., AND THALMANN, D. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface '88*, Canadian Information Processing Society, 26–33.

MARTHINSEN, A. 1999. Interpolation in Lie groups. *SIAM J. Numer. Anal. 37*, 1, 269–285.

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

MERRY, B., MARAIS, P., AND GAIN, J. 2006. Animation space: A truly linear framework for character animation. *ACM Trans. Graph. 25*, 4, 1400–1423.

MERRY, B., 2007. Personal communication.

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., AND HODGINS, J. K. 2006. Capturing and animating skin deformation in human motion. *ACM Trans. Graph. 25*, 3, 881–889.

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.

PRATSCHER, M., COLEMAN, P., LASZLO, J., AND SINGH, K. 2005. Outside-in anatomy based character rigging. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 329–338.

RHEE, T., LEWIS, J., AND NEUMANN, U. 2006. Real-time weighted pose-space deformation on the GPU. *Computer Graphics Forum 25*, 3, 439–448.

SCHEEPERS, F., PARENT, R. E., CARLSON, W. E., AND MAY, S. F. 1997. Anatomy-based modeling of the human musculature. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 163–172.

SHOEMAKE, K., AND DUFF, T. 1992. Matrix animation and polar decomposition. In *GI '92*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 258–264.

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.

SHREINER, D., WOO, M., NEIDER, J., AND DAVIS, T. 2007. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2.1.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

SLOAN, P.-P. J., ROSE, III, C. F., AND COHEN, M. F. 2001. Shape by example. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press, 135–143.

TERAN, J., SIFAKIS, E., BLEMKER, S. S., NG-THOW-HING, V., LAU, C., AND FEDKIW, R. 2005. Creating and simulating skeletal muscle from the visible human data set. *IEEE Transactions on Visualization and Computer Graphics 11*, 3, 317–328.

WANG, X. C., AND PHILLIPS, C. 2002. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, 129–138.

WANG, R. Y., PULLI, K., AND POPOVIĆ, J. 2007. Real-time enveloping with rotational regression. *ACM Trans. Graph. 26*, 3, 73.

WANG, W., JÜTTLER, B., ZHENG, D., AND LIU, Y. 2008. Computation of rotation minimizing frames. *ACM Trans. Graph. 27*, 1, 1–18.

WAREHAM, R., CAMERON, J., AND LASENBY, J. 2005. Applications of conformal geometric algebra in computer vision and graphics. *Lecture Notes in Computer Science 3519*, 329–349.

WEBER, O., SORKINE, O., LIPMAN, Y., AND GOTSMAN, C. 2007. Context-aware skeletal shape deformation. *Computer Graphics Forum (Proceedings of Eurographics) 26*, 3.

YANG, X., SOMASEKHARAN, A., AND ZHANG, J. J. 2006. Curve skeleton skinning for human and creature characters: Research articles. *Comput. Animat. Virtual Worlds 17*, 3-4, 281–292.