Skinning with Dual Quaternions

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

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

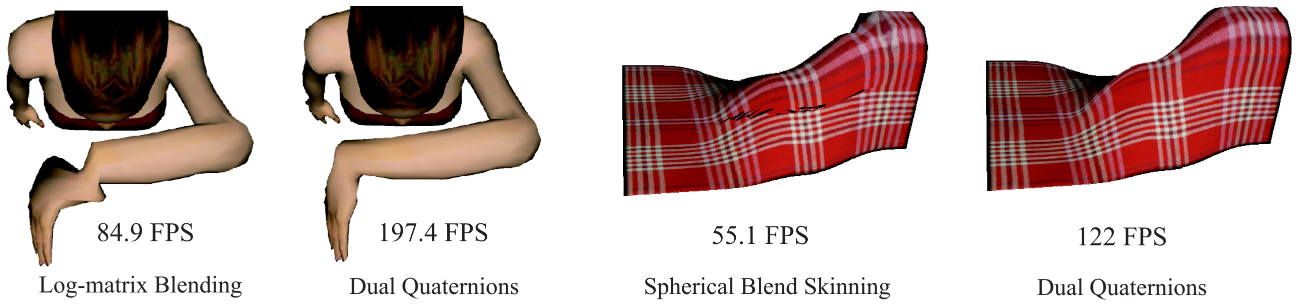

Figure 1: A comparison of dual quaternion skinning with previous methods: log-matrix blending [Cordier and Magnenat-Thalmann 2005] and spherical blend skinning [Kavan and Zara 2005]. The proposed approach not only eliminates artifacts, but is also much easier to implement and more than twice as fast.

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 GPU-friendly skinning algorithm based on dual quaternions. We show that this approach solves the artifacts of linear blend skinning at minimal additional cost. Upgrading an existing animation system (e.g., in a videogame) from linear to dual quaternion skinning is very easy and has negligible 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 – especially in the context of videogames.

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 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 matrices [Hejl 2004; Kavan and Zara 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 Zara's method [2005] does not have this restriction, but uses a complex and computationally expensive Singular Value Decomposition scheme. Obviously, the elegancy 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 the group of rigid transformations. Nothing prevents us from blending, for example, 3-tuples $(axis, angle, translation)$ or pairs $(axis + translation, \sin(angle))$. Even if we restrict ourselves to blending via linear combinations, we can construct infinitely many different blending methods just by considering different parameterizations of rigid transformations.

A natural question arises: which parameterization of rigid transformations is the best one for skinning? First of all, it is necessary to clarify what "best" means in this context. We summarize the mathematical properties that an ideal blending method for skinning should possess in Section 3.1. Next, we show that these properties are satisfied by a representation of rigid transformations known as dual quaternions – a generalization of regular quaternions invented in the nineteenth century [Clifford 1882].

From a practical point of view, dual quaternions offer a very simple yet efficient skinning method. Due to the properties of dual

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

quaternion blending, none of the skin collapsing effects exhibited by linear blend skinning 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.

2 Related Work

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.

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 only to existing subjects.

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], correction by principal components of example deformations [Kry et al. 2002] or fit the linear blending parameters to match the provided examples [Wang and Phillips 2002; Mohr and Gleicher 2003]. Recently, 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]. Generally, this class of methods 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 memory to store them and the animator's labour.

Geometric methods. In this case, only one input mesh is given (designed in a reference pose). The skeleton-to-skin binding is defined in a direct, geometrical way. The most popular way, 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 each influencing joint must be specified. Advanced blending methods, e.g., direct quaternion blending [Hejl 2004], log-matrix blending [Cordier and Magnenat-Thalmann 2005] and spherical blending [Kavan and Zara 2005] also use this rigging structure. Even though these techniques remove some artifacts, they still fall short of delivering natural skin deformation in all postures (see Figure 1 and Section 3.1).

Some researchers propose combatting skinning artifacts by implementing a different rigging method, e.g., based on sweep surfaces [Hyun et al. 2005] or auxiliary curved skeletons [Yang et al. 2006; Forstmann and Ohya 2006]. In some cases, this also allows advanced effects to be animated, such as muscle bulging. The disadvantages include complexity of the GPU implementation (even the very efficient Forstmann's method [2006] is about 3 times slower than linear blend skinning) and inconsistency with the established rigging standard: new rigging tools and data formats are needed. In this paper, we argue that the problems of linear blending do not stem from incorrect rigging, but from incorrect blending. With dual quaternion skinning, it is therefore neither necessary to change the rigging structures nor to update existing models.

Dual quaternions. The first reference to dual quaternions (historically called bi-quaternions) appears in [Clifford 1882], along with 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]. 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.

3 Dual Quaternion Skinning

In this section, we first quickly review previous geometric skinning algorithms (Section 3.1), identifying their strengths and weaknesses and deriving properties that an ideal blending for skinning should possess. Then we present a short tutorial on dual quaternions (Section 3.2) and discuss dual quaternion blending (Section 3.3). The final dual quaternion skinning algorithm is summarized in Section 3.4.

Conventions. We denote scalars by lower-case letters, vectors and quaternions in bold and matrices by capital letters. Dual quantities are distinguished from non-dual by a caret, e.g., $\hat{a}$ 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 $\mathbf{v} = (v_1, \dots, v_n)$. 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. Cross product is denoted as $\mathbf{v} \times \mathbf{w}$.

3.1 Background on Geometric Skinning

In the following section, we focus only on geometric skinning methods with linear blend skinning-style rigging structure (which is a 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 designed in a reference pose). The nodes of the skeleton represent joints and the edges can be interpreted as bones. However, each node (except the root node) can be easily identified with the edge leading to it, so the difference between joints and bones is rather moot in our case. The transformations relating joints in the hierarchy are assumed to be rigid (we do not consider scale or shear). 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. We usually store the joints in an array, with every joint referenced by a number $1,\dots,p$. In the reference (rest) pose, each joint has an associated local coordinate system. The transformation from the rest-pose of joint j to its actual position in the animated posture can be expressed by a rigid transformation matrix – let us denote this matrix as C_j.

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, following from graphics hardware considerations. The weights are normally assumed to be convex, i.e., $w_i \geq 0$ and $\sum_{i=1}^{n} w_i = 1$ (however, this non-negativity is not exploited in our algorithm). The i-th component of vector $\mathbf{w}$, w_i, represents the amount of influence of joint j_i on vertex $\mathbf{v}$.

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

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

that is, taking a linear combination of joint transformation matrices. This explains the skin collapsing artifacts (see Figure 2) – the blended matrix $\sum_{i=1}^{n} w_i C_{j_i}$ is no longer a rigid transformation, but a general affine one (potentially containing scale and shear factors). Formally speaking, the skin collapsing effects visualize the fact that the set of orthonormal matrices is not closed under addition.

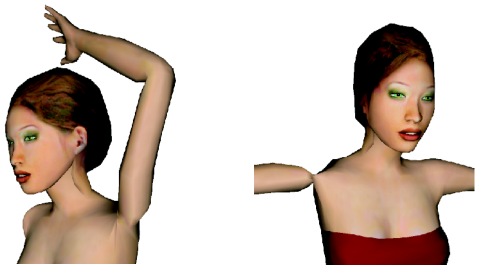
Figure 2: Typical "candy-wrapper" artifacts of linear blend skinning.

A blending method that avoids such skin-collapsing defects must therefore deliver a *rigid* transformation in all cases. The blending of matrix logarithms [Alexa 2002] satisfies this condition and has been implemented for skinning by [Cordier and Magnenat-Thalmann 2005]. Unfortunately, log-matrix blending has a different problem: in some cases, it picks a longer trajectory than necessary when interpolating rotation [Bloom et al. 2004]. This means that instead of a direct, straight rotation, a diversion is sometimes taken, which can result in unsightly artifacts (see Figure 3). Our desired blending should therefore always interpolate rigid transformations along the shortest path, i.e., along a geodesic in the manifold of rigid transformations [Hofer and Pottmann 2004]. Note that this can be achieved for log-matrix blending in the case of two transformations [Li and Hao 2006], but this method does not generalize to n transformations.

Decomposing rigid transformation matrices into $(quaternion, translation)$ pairs and blending them linearly

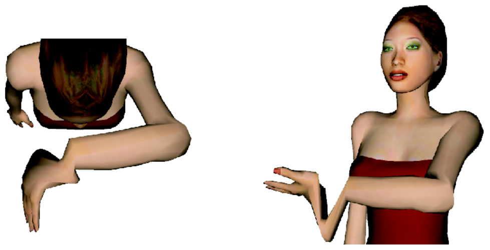
Figure 3: Artifacts of log-matrix blending [Alexa 2002] caused by choosing a longer trajectory than necessary.

has none of the above-mentioned problems: it always produces rigid transformations and selects the shortest path of the interpolated rotation [Kavan and Zara 2005]. Unfortunately, the blending of quaternions and translation vectors independently has another pitfall – dependence on the body-space coordinate system. In practice, this means that the model's vertices rotate around the origin of the body-space. The origin is user-defined and usually coincides with the center of mass. In many applications, rotation around the center of mass is exactly what we need. However, this is not the case for skinning.

Rotation of model vertices around the center of mass produces unacceptable skin deformations. Intuitively, vertices influenced by both the lower and upper arm should naturally rotate about a point near the elbow. This is the principle of [Hejl 2004]: the vertices rotate around the nearest joint. In cases similar to that of the elbow (two bones connected by a joint), this works perfectly. However, for more complex joint influences, such as those usually occurring around the armpit or dorsum, artifacts result from the fact that the center of rotation is fixed and does not adapt to the actual posture. In character models, these artifacts can be usually avoided by careful rigging. However, they are inevitable in other deformable objects, such as our simple cloth model shown in Figure 4.

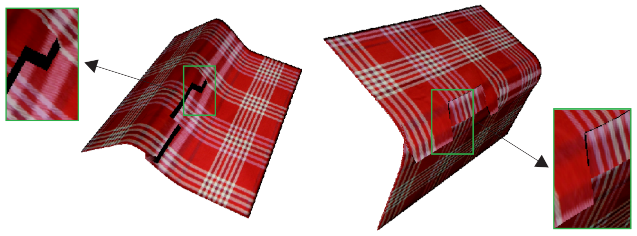
Figure 4: Artifacts of direct quaternion blending [Hejl 2004] caused by fixed rotation centers.

The limitations of Hejl's method can be resolved as described in [Kavan and Zara 2005], which presents an accurate method to compute the rotation centers. It works by minimizing the translation of the resulting blended transformation (c.f. with Hejl's method which works correctly only when this translation is zero). Unfortunately, the cost of this approach is significant: it requires a least-squares solution of a set of equations, carried out using Singular Value Decomposition (SVD). Since it is not affordable to run SVD per vertex, the rotation centers are cached and reused for clusters of vertices. This trick enables real-time performance of spherical blend skinning to be achieved. Unfortunately, the recycling of rotation centers may cause discontinuities of the deformed skin in areas where the clusters concur (see Figure 5). These artifacts are however much less frequent (and apparent) than with direct quaternion blending.

Note that it was not necessary (and not even possible) to choose a rotation center in linear or log-matrix blending. This is because both of these approaches are coordinate invariant, i.e., it does not matter if we switch our transformations to a new coor-

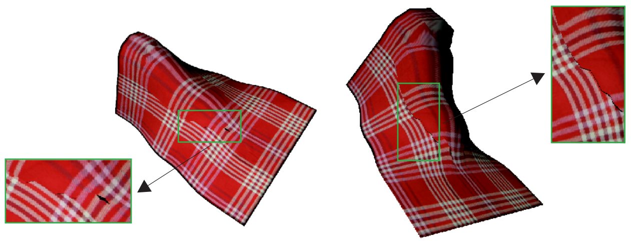

Figure 5: Discontinuities of spherical blend skinning [Kavan and Zara 2005] caused by caching rotation centers.

dinate system before or after the blending. This is easy to verify: if matrix M represents a change of coordinates, then matrices $C_{j_1},\ldots,C_{j_n}$ expressed with respect to the new coordinate system become $MC_{j_1}M^{-1},\ldots,MC_{j_n}M^{-1}$. Linear blending with these matrices gives:

$$\left(\sum_{i=1}^{n} w_i M C_{j_i} M^{-1}\right) = M\left(\sum_{i=1}^{n} w_i C_{j_i}\right) M^{-1}$$

which follows from the distributive property of matrix multiplication. We see that the resulting blended matrix is really the same as that for matrices $C_{j_1},\ldots,C_{j_n}$, just with respect to the new coordinate system (of course). The same reasoning can be repeated for log-matrix blending if we recall the following properties of matrix exponential and logarithm: $\exp(MC_{j_i}M^{-1}) = M\exp(C_{j_i})M^{-1}$ and $\log(MC_{j_i}M^{-1}) = M\log(C_{j_i})M^{-1}$ (see [Moakher 2002]). This shows that log-matrix blending is coordinate-invariant.

Why does this not work for $(quaternion, translation)$ pairs? The reason is that if we define an algebra over $(quaternion, translation)$ pairs, with multiplication corresponding to composition of transformations, then this algebra is not distributive. Specifically, it is not distributive with respect to right multiplication [Kavan and Zara 2005] (it is with respect to left multiplication). As right multiplication corresponds to the change of body-space coordinates (and left multiplication to the change of world-space coordinates), this just reflects the fact that blending $(quaternion, translation)$ pairs rotates about the origin of the body-space.

To summarize, we see that an optimal rigid transformation blending method for skinning should:

- always return a valid rigid transformation (to prevent skin-collapsing effects)
- be coordinate-invariant (to avoid the problems with rotation center)
- interpolate two rigid transformations along the shortest path (to mimic natural skin behaviour and avoid excessive stretching)

Note that because we work with all types of rigid transformations, the shortest path property actually embraces two properties: shortest path rotation and translation. Shortest path rigid transformations correspond to *screws* (see next section). Besides the above stated properties, the blending should of course be efficiently computable, typically on a GPU. As we will show in the remainder of this paper, our proposed dual quaternion blending satisfies all of these requirements.

3.2 Dual Quaternions

Dual quaternions are not a standard tool in computer graphics, in contrast to regular quaternions. This section provides a brief tutorial; for further details, see [McCarthy 1990; Kavan et al. 2006]. 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. The algebra of dual numbers is similar to complex numbers: any dual number $\hat{a}$ can be written as $\hat{a} = a_0 + \varepsilon a_\varepsilon$, where a_0 is the non-dual part, a_ε the dual part and ε is a *dual unit* satisfying $\varepsilon^2 = 0$. The dual conjugate is analogous to the complex conjugate: $\overline{\hat{a}} = a_0 - \varepsilon a_\varepsilon$. Multiplication of two dual numbers is given as $(a_0 + \varepsilon a_\varepsilon)(b_0 + \varepsilon b_\varepsilon) = a_0 b_0 + \varepsilon(a_0 b_\varepsilon + a_\varepsilon b_0)$. The inverse of a dual number $\hat{a}^{-1}$ is given by

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

as can be immediately verified. The previous expression is defined only when $a_0 \neq 0$. Purely dual numbers, that is dual numbers with $a_0 = 0$, do not have an inverse. This is a fundamental difference from complex numbers, because every non-zero complex number has an inverse. The square root is defined only for dual numbers with a positive non-dual part, and it is computed as

$$\sqrt{a_0 + \varepsilon a_\varepsilon} = \sqrt{a_0} + \varepsilon\frac{a_\varepsilon}{2\sqrt{a_0}}$$

A dual quaternion $\hat{\mathbf{q}}$ can be written as $\hat{\mathbf{q}} = \hat{w} + i\hat{x} + j\hat{y} + k\hat{z}$, where $\hat{w}$ is the scalar part (dual number), $(\hat{x},\hat{y},\hat{z})$ is the vector part (dual vector), and i,j,k are the usual quaternion units. The dual unit ε commutes with quaternion units, for example $i\varepsilon = \varepsilon i$. A dual quaternion can 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: $\hat{\mathbf{q}}^* = \mathbf{q}_0^* + \varepsilon\mathbf{q}_\varepsilon^*$. The norm of a dual quaternion can be written as $\|\hat{\mathbf{q}}\| = \sqrt{\hat{\mathbf{q}}^*\hat{\mathbf{q}}} = \sqrt{\hat{\mathbf{q}}\hat{\mathbf{q}}^*}$, which expands to

$$\|\hat{\mathbf{q}}\| = \sqrt{\hat{\mathbf{q}}^*\hat{\mathbf{q}}} = \|\mathbf{q}_0\| + \varepsilon\frac{\langle\mathbf{q}_0,\mathbf{q}_\varepsilon\rangle}{\|\mathbf{q}_0\|} \qquad (3)$$

The norm satisfies the usual property $\|\hat{\mathbf{p}}\hat{\mathbf{q}}\| = \|\hat{\mathbf{p}}\|\|\hat{\mathbf{q}}\|$. The inverse of a dual quaternion is defined only when $\mathbf{q}_0 \neq \mathbf{0}$. In this case, we have $\hat{\mathbf{q}}^{-1} = \frac{\hat{\mathbf{q}}^*}{\|\hat{\mathbf{q}}\|^2}$. Unit dual quaternions are those satisfying $\|\hat{\mathbf{q}}\| = 1$. According to the previous formula, a dual quaternion $\hat{\mathbf{q}}$ is unit if and only if $\|\mathbf{q}_0\| = 1$ and $\langle\mathbf{q}_0,\mathbf{q}_\varepsilon\rangle = 0$. Note that unit dual quaternions are always invertible (their inverse is just conjugation). Just like ordinary quaternions, dual quaternions are also associative, distributive, but not commutative.

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

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

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

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

$$(1 + \frac{\varepsilon}{2}(t_0 i + t_1 j + t_2 k))\mathbf{q}_0 = \mathbf{q}_0 + \frac{\varepsilon}{2}(t_0 i + t_1 j + t_2 k)\mathbf{q}_0 \qquad (4)$$

We can verify, by direct computation, that the result is always a unit dual quaternion and, conversely, that any unit dual quaternion can always be written in this form. 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 (4) 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 $2\mathbf{q}_\varepsilon \mathbf{q}_0^*$.

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

$$\hat{\mathbf{q}} = \cos\frac{\hat{\theta}}{2} + \hat{\mathbf{s}}\sin\frac{\hat{\theta}}{2} \qquad (5)$$

where $\hat{\mathbf{s}}$ is a unit dual vector with zero scalar part, see [McCarthy 1990] or [Daniilidis 1999]. Note that this looks like the formula for regular quaternions, just employing the dual angle $\hat{\theta} = \theta_0 + \varepsilon\theta_\varepsilon$ and unit dual vector $\hat{\mathbf{s}} = \mathbf{s}_0 + \varepsilon\mathbf{s}_\varepsilon$. The geometric interpretation of those quantities is related to *screw motion*, that is a rotation and translation about the same axis. Chasle's theorem [Daniilidis 1999] states that any rigid transformation can be described by a screw motion (see Figure 6). Angle $\theta_0/2$ is the angle of rotation, and unit vector $\mathbf{s}_0$ represents the direction of the axis of rotation. $\theta_\varepsilon/2$ is the amount of translation along vector $\mathbf{s}_0$, and $\mathbf{s}_\varepsilon$ is the *moment* of the axis. Moment is an unambiguous description of the position of an axis in space. It is given by equation $\mathbf{s}_\varepsilon = \mathbf{p} \times \mathbf{s}_0$, where $\mathbf{p}$ is a vector pointing from the origin to an arbitrary point on the axis. Which point we choose is not important, because for any other point of the axis, say $\mathbf{p} + c\mathbf{s}_0$ (where c is an arbitrary scalar), we obtain the same moment: $(\mathbf{p} + c\mathbf{s}_0) \times \mathbf{s}_0 = \mathbf{p} \times \mathbf{s}_0$. In other words, we can say that while classical quaternions can represent only rotations whose axes pass through the origin, dual quaternions can represent rotations with arbitrary axes.

Dual quaternions exhibit the so-called *antipodal* property of classical quaternions, i.e., the fact that both $\hat{\mathbf{q}}$ and $-\hat{\mathbf{q}}$ represent the same rigid transformation. Therefore, when converting matrices to dual quaternions, we must choose an appropriate sign. This is however the same problem as in the case of regular quaternions, but we can apply the same methods as in [Hejl 2004] or [Kavan and Zara 2005], i.e., consistently select signs so that the resulting quaternions lie in the same hemisphere.

3.3 Blending of Dual Quaternions

Using Equation (4), we can convert our skinning transformations to unit dual quaternions $\hat{\mathbf{q}}_1, \ldots, \hat{\mathbf{q}}_n$. The next task is to compute a blended unit dual quaternion $\hat{\mathbf{q}}$ with respect to the given convex weights $\mathbf{w} = (w_1, \ldots, w_n)$. This can be done by taking their linear combination followed by a normalization (because we need a *unit* dual quaternion). We call this Dual quaternion Linear Blending (DLB):

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

In the following, we show that DLB has all the properties required in skinning (as summarized in Section 3.1).

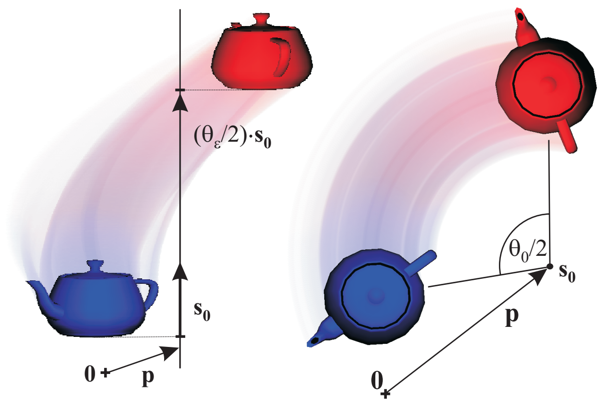

Figure 6: Example of a screw motion, side and top view.

Firstly, it is clear that *DLB* always returns a rigid transformation, because *DLB* computes a unit dual quaternion which can be subsequently converted to a rigid transformation matrix.

Secondly, in order to demonstrate the coordinate-invariance of *DLB*, we have to verify that for any unit dual quaternion $\hat{\mathbf{r}}$, the following formula is true:

$$DLB(\mathbf{w}; \hat{\mathbf{r}}\hat{\mathbf{q}}_1\hat{\mathbf{r}}^*, \ldots, \hat{\mathbf{r}}\hat{\mathbf{q}}_n\hat{\mathbf{r}}^*) = \hat{\mathbf{r}}DLB(\mathbf{w}; \hat{\mathbf{q}}_1, \ldots, \hat{\mathbf{q}}_n)\hat{\mathbf{r}}^* \qquad (6)$$

In fact, this breaks down to verifying two similar properties, called left and right invariance. Left invariance requires that

$$DLB(\mathbf{w}; \hat{\mathbf{r}}\hat{\mathbf{q}}_1, \ldots, \hat{\mathbf{r}}\hat{\mathbf{q}}_n) = \hat{\mathbf{r}}DLB(\mathbf{w}; \hat{\mathbf{q}}_1, \ldots, \hat{\mathbf{q}}_n)$$

and right invariance, by analogy, requires that:

$$DLB(\mathbf{w}; \hat{\mathbf{q}}_1\hat{\mathbf{r}}, \ldots, \hat{\mathbf{q}}_n\hat{\mathbf{r}}) = DLB(\mathbf{w}; \hat{\mathbf{q}}_1, \ldots, \hat{\mathbf{q}}_n)\hat{\mathbf{r}}$$

It should be clear that left and right invariance together imply coordinate-invariance. To prove left invariance, it is sufficient to apply the distributive property of dual quaternions and the multiplicative property of the norm (Section 3.2):

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

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

because $\|\hat{\mathbf{r}}\| = 1$, according to our assumption. Demonstration of the right invariance proceeds along the same lines. We see that *DLB* is indeed coordinate-invariant.

Thirdly, we show that when *DLB* is applied to two rigid transformations, it interpolates them along the shortest trajectory. Let therefore $\hat{\mathbf{p}}, \hat{\mathbf{q}}$ be two unit dual quaternions. We will denote their blending as $DLB(t; \hat{\mathbf{p}}, \hat{\mathbf{q}})$. Thanks to the left invariance of *DLB* (proven above) we can write

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

It is therefore sufficient to show that the path between 1 (the identity transformation) and $\hat{\mathbf{p}}^*\hat{\mathbf{q}}$ will be the shortest one, i.e., given by the screw corresponding to $\hat{\mathbf{p}}^*\hat{\mathbf{q}}$. Since $\hat{\mathbf{p}}, \hat{\mathbf{q}}$ are unit dual quaternions, so is $\hat{\mathbf{p}}^*\hat{\mathbf{q}}$ and therefore, according to Equation (5), there exists $\hat{\mathbf{n}}$ (a unit dual quaternion with zero scalar part) and $\hat{\alpha}$ (a dual scalar) such that $\hat{\mathbf{p}}^*\hat{\mathbf{q}} = \cos\frac{\hat{\alpha}}{2} + \hat{\mathbf{n}}\sin\frac{\hat{\alpha}}{2}$. Therefore, $DLB(t; 1, \hat{\mathbf{p}}^*\hat{\mathbf{q}})$ can be re-written as

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

which means that the screw axis of $DLB(t;1,\hat{\mathbf{p}}^*\hat{\mathbf{q}})$ is the same for all $t \in [0,1]$ and is given by $\hat{\mathbf{n}}$ – the only thing that varies is the angle of rotation and amount of translation, both encoded in $\frac{1-t+t\cos(\frac{\hat{\alpha}}{2})}{\|1-t+t\hat{\mathbf{p}}^*\hat{\mathbf{q}}\|}$ and $\frac{t\sin(\frac{\hat{\alpha}}{2})}{\|1-t+t\hat{\mathbf{p}}^*\hat{\mathbf{q}}\|}$. In other words, $DLB(t;1,\hat{\mathbf{p}}^*\hat{\mathbf{q}})$ produces a shortest path screw motion. It should be noted that this screw motion does not have a constant speed – in an analogy to linear interpolation of regular quaternions. However, it can be demonstrated that the velocity of this motion is actually not far from constant [Kavan et al. 2006], which explains why this issue does not present any visible drawbacks in skinning.

3.4 The Final Algorithm

This section discusses how to efficiently implement dual quaternion skinning. First, we need to convert the skinning matrices $C_1,\dots,C_p$ (where p is the total number of joints) to dual quaternions $\hat{\mathbf{q}}_1,\dots,\hat{\mathbf{q}}_p$, unless our application works with them already. This will typically be done on a CPU and will not take long, because the conversion to a dual quaternion involves just one quaternion multiplication (see Equation (4)) and the number of joints p is usually quite small. The dual quaternions $\hat{\mathbf{q}}_1,\dots,\hat{\mathbf{q}}_p$ 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 conversion back to the matrix form, takes place in the vertex shader. Since the number of vertices tends to be orders of magnitude higher than the number of joints, the vertex shader code should be optimized. The first step of $DLB(\mathbf{w};\hat{\mathbf{q}}_{j_1},\dots,\hat{\mathbf{q}}_{j_n})$, i.e., the computation of the linear combination $\hat{\mathbf{b}} = \sum_{i=1}^n w_i\hat{\mathbf{q}}_{j_i}$, is straightforward. However, the following normalization, i.e., the computation of $\hat{\mathbf{b}}' = \hat{\mathbf{b}}/\|\hat{\mathbf{b}}\|$, and the subsequent conversion of $\hat{\mathbf{b}}'$ to matrix M can be optimized as follows:

If $\hat{\mathbf{b}} = \mathbf{b}_0 + \varepsilon\mathbf{b}_\varepsilon$, then the norm $\|\hat{\mathbf{b}}\| = \|\mathbf{b}_0\| + \varepsilon\frac{\langle\mathbf{b}_0,\mathbf{b}_\varepsilon\rangle}{\|\mathbf{b}_0\|}$, according to Equation (3). 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 (2). Therefore,

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

Now, $\hat{\mathbf{b}}' = \mathbf{b}'_0 + \varepsilon\mathbf{b}'_\varepsilon$ needs to be converted to a homogeneous matrix M. Its rotational part is given simply by $\mathbf{b}'_0 = \frac{\mathbf{b}_0}{\|\mathbf{b}_0\|}$. The translation is given by the vector part of $2\mathbf{b}'_\varepsilon(\mathbf{b}'_0)^*$. This simplifies 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 $2\mathbf{b}'_\varepsilon(\mathbf{b}'_0)^* = 0$ (according to Equation (4)), it means that there is no need to evaluate $\frac{\langle\mathbf{b}_0,\mathbf{b}_\varepsilon\rangle}{\|\mathbf{b}_0\|^2}$, because its purpose is only to cancel out the scalar part of $\frac{\mathbf{b}_\varepsilon\mathbf{b}_0^*}{\|\mathbf{b}_0\|^2}$. Therefore, we can compute the translational part of matrix M just by computing the vector part of $2\frac{\mathbf{b}_\varepsilon\mathbf{b}_0^*}{\|\mathbf{b}_0\|^2}$ (and its scalar part can be safely ignored). This means that we can avoid the lengthy dual quaternion normalization.

Matrix M is then used to transform the input vertex $\mathbf{v}$. Thanks to the fact that M is always a rigid transformation matrix, there are no complications with the inverse transposition and normalization (as in linear blending [Mohr and Gleicher 2003]) and we can use M directly to transform also the vertex normal $\mathbf{v}_n$. Dual quaternion skinning can thus be summarized in the following pseudocode:

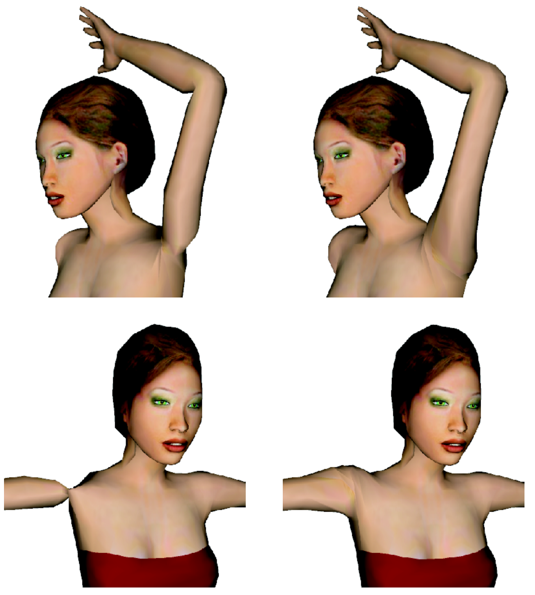

Figure 7: Comparison of linear (left) and dual quaternion blending (right). Dual quaternions preserve rigidity of input transformations and therefore avoid skin collapsing artifacts.

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

    $\hat{\mathbf{b}} = w_1\hat{\mathbf{q}}_{j_1} + \dots + w_n\hat{\mathbf{q}}_{j_n}$
    // 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 components of $\mathbf{c}_0$ as $w_0,x_0,y_0,z_0$
    // denote the components of $\mathbf{c}_\varepsilon$ as $w_\varepsilon,x_\varepsilon,y_\varepsilon,z_\varepsilon$
    $t_0 = 2(-w_\varepsilon x_0 + x_\varepsilon w_0 - y_\varepsilon z_0 + z_\varepsilon y_0)$
    $t_1 = 2(-w_\varepsilon y_0 + x_\varepsilon z_0 + y_\varepsilon w_0 - z_\varepsilon x_0)$
    $t_2 = 2(-w_\varepsilon z_0 - x_\varepsilon y_0 + y_\varepsilon x_0 + z_\varepsilon w_0)$
    $M = \begin{pmatrix} 1-2y_0^2-2z_0^2 & 2x_0y_0-2w_0z_0 & 2x_0z_0+2w_0y_0 & t_0 \\ 2x_0y_0+2w_0z_0 & 1-2x_0^2-2z_0^2 & 2y_0z_0-2w_0x_0 & t_1 \\ 2x_0z_0-2w_0y_0 & 2y_0z_0+2w_0x_0 & 1-2x_0^2-2y_0^2 & t_2 \end{pmatrix}$
    $\mathbf{v}' = M\mathbf{v}$ // where $\mathbf{v}$ has form $\mathbf{v} = (v_0,v_1,v_2,1)$
    $\mathbf{v}'_n = M\mathbf{v}_n$ // where $\mathbf{v}_n$ has form $\mathbf{v}_n = (v_{n,0},v_{n,1},v_{n,2},0)$
```

The actual vertex shader code would add some lighting computations and transformation by the model-view-projection matrix. Note that the formulas for t_0,t_1,t_2 are simply the expanded forms of quaternion product $2\mathbf{c}_\varepsilon\mathbf{c}_0^*$. It is encouraging how simple the resulting algorithm is.

4 Results and Comparison

In our experiments, we use a female model with 5002 vertices, 9253 triangles and 54 joints. We 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 Zara 2005]. 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

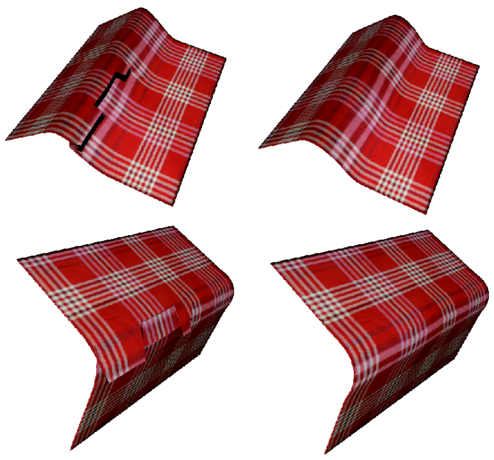

Figure 8: Comparison of direct quaternion blending (left) and dual quaternion blending (right). Only the latter delivers smooth deformation.

support our hypothesis that our DLB method (which satisfies the properties stated in Section 3.1) is indeed free of artifacts, see Figures 7, 8, 9 and 10.

In order to compare computational performance, we have implemented both CPU and GPU versions of dual quaternion skinning. The average times are reported in Figure 11. The performance of log-matrix vs. spherical blending has been compared only on the CPU, because the GPU implementation of these algorithms has not been provided in the previous work. Note that our implementation of log-matrix blending uses an optimization for rigid transformations based on the Rodrigues formula, as suggested in [Alexa 2002].

From the measurements, we see that dual quaternion, linear and direct quaternion blending [Hejl 2004] have quite similar performance. 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).

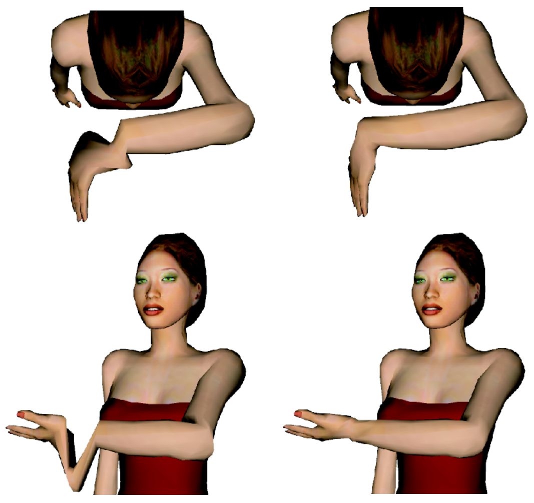

Figure 9: Comparison of log-matrix (left) and dual quaternion blending (right). The shortest-path property of dual quaternion blending guarantees natural skin deformations.

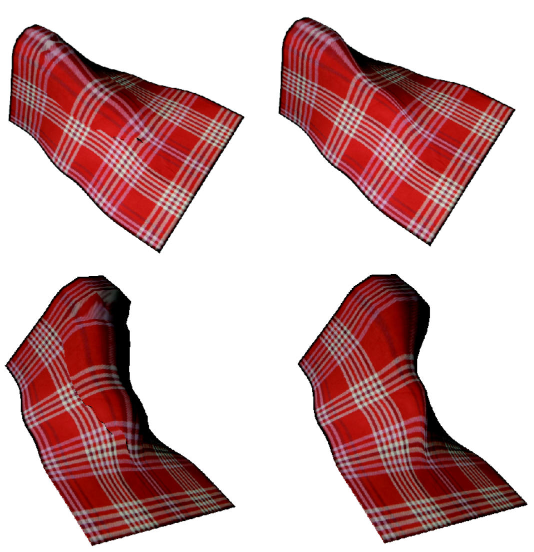

Figure 10: Comparison of spherical blending (left) and dual quaternion blending (right). Dual quaternions do not need to cluster vertices and therefore naturally avoid artifacts.

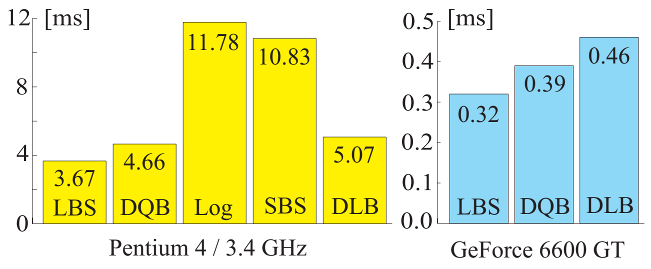

Figure 11: Average CPU/GPU runtimes for skin deformation of the woman model in milliseconds: LBS – linear blend skinning, DQB – direct quaternion blending, Log – log-matrix blending, SBS – spherical blend skinning, DLB – dual quaternion linear blending.

5 Conclusions

In this paper, we propose a novel skinning method based on the blending of dual quaternions. We justify its usefulness not only on practical examples, but also by identifying the mathematical properties important for skinning and showing that dual quaternions satisfy them. Our method is efficient, very simple to implement and does not require the modification of existing models or authoring tools (even though advanced methods exploiting the linearity of linear blend skinning, such as [Mohr and Gleicher 2003], would have to be adapted). We therefore believe that it provides a really practical alternative to the popular but inaccurate linear blend skinning.

The proposed algorithm has several limitations. Dual quaternion skinning simulates skin-to-skeleton binding in a simplistic way and is therefore unable to produce realistic musculature or dynamic effects. Also, dual quaternions are limited only to rigid transformations and are thus not suitable for models whose parts can scale or shear (e.g., some cartoon characters).

From a broader point of view, the blending of dual quaternions has potentially many more applications than just skinning. More generally, it is a method for rigid transformation blending with interesting properties. Therefore, it could be potentially 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 Acknowledgements

We wish to thank Carlo H. Séquin for early illuminating discussion on the topic and the anonymous reviewers for their helpful comments. We acknowledge the support of the Higher Education Authority of 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.

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.

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`.

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.

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.

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.

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

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

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

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.

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.

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.

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.

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

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.

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

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.

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.

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.