Temporal Face Normal Interpolation

Jindřich Parus [*]
Centre of Computer Graphics
and Data Visualization,
University of West Bohemia,
Pilsen, Czech Republic

Anders Hast [†]
Creative Media Lab,
University of Gävle,
Gävle, Sweden

Ivana Kolingerová [‡]
Centre of Computer Graphics
and Data Visualization,
University of West Bohemia,
Pilsen, Czech Republic

Abstract

Normals of triangular faces are essential vectors in many areas of computer graphics. In this paper we will deal with methods for normal computation of triangles under linear soft-body deformation, i.e., the triangles which deform in time so that each vertex travels independently along its linear trajectory. Linear deformation can be found in mesh morphing, cloth simulation, physical simulation, etc. We will demonstrate five different approaches for temporal face normal interpolation, one of them is new, and we will discuss their pros and cons.

1 Introduction

Normal vectors are important vectors in many areas of computer graphics. For example in shading, they are used for light model evaluation [Phong 1975][Gouraud 1971]. In rendering they are used for back face culling to exclude primitives, which can not be visible, from the rendering pipeline and thereby reducing the amount of primitives to be sent to the graphics pipeline. In computational geometry they are used, e.g., for point containment queries, to distinguish between the interior and exterior of objects. In collision detection they are used for example for collision response computation [Eberly 2004].

A normal vector **N** at a point **P** of the surface S is a vector perpendicular to the surface *S* at point **P**. In computer graphics we usually work with some approximation of the real surface. Due to its simplicity a piecewise linear approximation is very often used to represent surfaces. Here a surface is described by a set of triangular faces. Such representation is called a mesh. It is a very widespread representation because it is very easy to modify, store and render since it is supported by graphics hardware.

In the case of meshes we usually do not define a normal in each point of the surface. Instead we define the normals of each piecewise linear segment, i.e., triangular face. Then it is supposed that the normal is constant over the whole face. The normal of the face is usually computed by taking the cross-product of two edges of the triangle. In fact we can use any two vectors which lay in the plane of the face and which are not parallel. Nonetheless since the triangular face is defined by three vertices, the easiest way to compute the face normal is to use two edges. Note that there are always two possible face normals depending on which order of vertices we choose for normal computation. It is usually required that all normals of the mesh are consistently oriented, which implies consistent orientation of faces, i.e., order of vertices which define the face. Face normals are used for example in flat shading, where a light model is evaluated only once for one face, then the entire face is filled with one single color. It is of course very fast but the final image may produce Mach bands, which are disturbing.

In many graphics applications it is required to compute normals for the vertices. A usual approach for computation of a vertex normal for the vertex **v** is to take a weighted average of normals of faces which are incident to the vertex **v**. There are several strategies for which weights to choose [Jin 2005] e.g., area of the faces, the angle between the edges at the vertex **v**, etc. Again, face normals are essential. Normals at vertices can then be interpolated over the face resulting in smooth normal variation, which is essential in shading. So for vertex normal computation it is required to compute the face normals as well.

Many problems in computer graphics have t-variant nature, i.e., data change in time. In the case of meshes, meshes may move in space, rotate, deform, etc. Once the mesh is modified it is necessary to update the normals as well to be able to carry out operations as described above with respect to a current state of the mesh. Usually we have some static version of a certain operation and when the state of the mesh is changed the operation is recomputed according to the new state of the mesh. It is simple but in some cases it may be very slow.

The normal computation methods for t-variant meshes can be categorized into two groups according to what kind of motion the mesh performs. The first group is rigid body-motion. In the rigid-body motion the relative position of any two vertices stays fixed during the transformation and the object transforms as one entity [Karni 2004]. Examples of rigid-body motion are rotation and translation. The second group is soft-body motion. In the soft-body motion there are no restrictions on change of relative position of any two vertices; each vertex can travel along its trajectory independently of the other vertices.

The rigid-body motion can be expressed by a transformation matrix **A**. Then the normal vectors of the transforming mesh under rigid-body motion is transformed by the inverse transpose of the Jacobian matrix **J** of the transformation matrix **A**, thus $(\mathbf{J^{-1}})^{\mathbf{T}}$ [Gomes 1999]. Moreover, if the transformation is linear, the normal field is transformed by the matrix **A** directly, because if the transformation represented by the matrix **A** is linear then it holds that $(\mathbf{J^{-1}})^{\mathbf{T}} = \mathbf{J} = \mathbf{A}$. In soft-body motion, as there are no restrictions on the motion, no global transformation can be applied on the normal vectors, as in the rigid-body motion case.

As in many computer graphics disciplines there is always discrepancy between fast computation and exact computation. Some time critical (real-time) applications require fast computation and some inaccuracies up to some level are excused, a typical example is shading. Other applications require the highest possible precision and may wait for an exact result if necessary, e.g., point containment tests are very critical and some inaccuracy in the result (inside/outside) may have large consequences.

In this paper we will deal with face normal computation of meshes under linear soft-body deformation, i.e., each vertex trav-

[*] e-mail: jparus@kiv.zcu.cz
[†] e-mail: aht@hig.se
[‡] e-mail: kolinger@kiv.zcu.cz

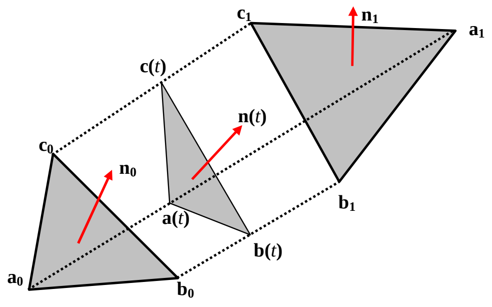


Figure 1: A Triangle deforms from the initial position $\mathbf{a_0}, \mathbf{b_0}, \mathbf{c_0}$ to the final position $\mathbf{a_1}, \mathbf{b_1}, \mathbf{c_1}$.

els independently along its linear trajectory. In section 2 we will describe five approaches for face normal interpolation. In section 3 we will compare described approaches from the time consumption and accuracy point of view and we will discuss pros and cons of each method. Section 4 concludes the paper together with some ideas for the future work.

2 Face Normal Interpolation

First let us describe more specifically our setting. We suppose that mesh is deforming in time so that only the position of vertices varies; the connectivity (i.e., interconnection of vertices by edges) is always the same. We know the current state of the mesh at time t_0 and we also know the future state of the mesh at time t_1, which is usually given by some simulation process (e.g., collision detection). An individual state of the mesh will be referred to as a keyframe. We suppose that the trajectory of vertices between individual keyframes is linear. More complicated curved trajectories can be approximated by piecewise linear curves and our approaches can be applied separately for individual linear segments of the trajectory. As the triangular mesh is just a set of triangles we will first consider only one moving triangular face. The setting is demonstrated in figure 1. A triangle deforms from the initial position $\mathbf{a_0}, \mathbf{b_0}, \mathbf{c_0}$ to the final position $\mathbf{a_1}, \mathbf{b_1}, \mathbf{c_1}$. The vertex trajectories $\mathbf{a}(t), \mathbf{b}(t), \mathbf{c}(t)$ are linear. We can easily compute the initial normal $\mathbf{n_0}$ (for the initial position of the face) and the final normal $\mathbf{n_1}$ (for the final position of the face). We are interested in how to compute the face normal $\mathbf{n}(t)$ when the face deforms between the initial position to the final position so that it more or less respects the triangle deformation.

The simplest approach is to recompute the face normal in each time instant by taking the cross-product of the edges, i.e.:

$$\mathbf{n} = (\mathbf{v_1} - \mathbf{v_0}) \times (\mathbf{v_2} - \mathbf{v_0}) \tag{1}$$

where $\mathbf{v_0}, \mathbf{v_1}, \mathbf{v_2}$ are vertices of triangle. The result is exact, but it requires a cross-product computation (9 additions, 6 multiplications) and normalization and therefore we are looking for other methods, which are perhaps less accurate but faster. Anyhow, they need to approximate the normal in a good way.

2.1 t-variant cross product

We can use the usual cross-product approach but instead of using vertex coordinates we use the trajectories of vertices [Parus 2006], i.e.:

$$\mathbf{n}(t) = (\mathbf{b}(t) - \mathbf{a}(t)) \times (\mathbf{c}(t) - \mathbf{a}(t)) \tag{2}$$

Since the vertex trajectories are linear then the result is a degree two polynomial for each component of the normal. If there are two trajectories with the same direction and the same length then equation (2) is degree one polynomial. If all three trajectories have the same direction and the same length then equation (2) is constant, i.e., not time dependent. By evaluating the degree two polynomials we obtain the exact normals. The advantage of this approach is that the coefficients of the degree two polynomials can be precomputed in a preprocessing stage. Moreover, the quadratic polynomials can be evaluated by the Horner scheme (6 additions, 6 multiplications). It should be noted that the computation can be reduced further as explained later. The length of the normal is proportional to the area of the face and therefore normalization is required. The normalization can also be included into a t-variant formula:

$$\mathbf{n}(t) = \frac{(\mathbf{b}(t) - \mathbf{a}(t)) \times (\mathbf{c}(t) - \mathbf{a}(t))}{\|(\mathbf{b}(t) - \mathbf{a}(t)) \times (\mathbf{c}(t) - \mathbf{a}(t))\|} \tag{3}$$

The resulting evaluation is thus more complicated.

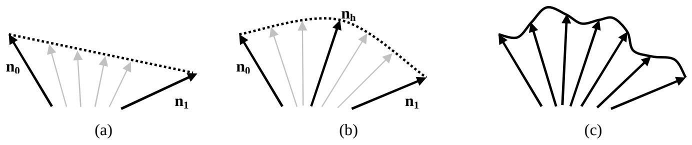


Figure 2: : (a) linear interpolation, (b) quadratic interpolation, (c) higher degree interpolation.

2.2 Lagrange interpolation

Linear interpolation (degree one Lagrange interpolation) of normal vectors is described by the following expression:

$$\mathbf{n}(t) = \mathbf{n_0} + t(\mathbf{n_1} - \mathbf{n_0}) \tag{4}$$

where $\mathbf{n_0}$ is an initial normal and $\mathbf{n_1}$ is the final normal. This approach is used for example in spatial normal interpolation in Phong shading. In fact, here, the vector is treated as if it was a point. It is fast but not sufficient because the intermediate normals are far from being perpendicular to the triangle; furthermore the intermediate normals need to renormalized figure 2(a). Higher degree Lagrange interpolation fits better the true behavior of the face normals but it is always a tradeoff between a better fit and an oscillation due to the higher degree of the interpolation function. Also, unit length is not preserved. It is demonstrated in figure 2. Figure 2 (a) shows linear normal interpolation, and it can be seen that intermediate normals (gray) do not have unit length. Figure 2 (b) shows degree two Lagrange interpolation which needs an additional intermediate normal $\mathbf{n_h}$ to fit a quadratic interpolation curve. Figure 2 (c) shows higher degree Lagrange interpolation where a number of intermediate normals are required to precompute the interpolation curve, it can be seen that due to the high degree of the interpolation polynomial the normal is oscillating.

2.3 Vector SLERP

SLERP (Spherical Linear intERPolation) [Glassner 1999] is a technique for interpolation of vectors, which maintains unit length, it is defined as:

$$SLERP(\mathbf{n_0}, \mathbf{n_1}, t) = \frac{\sin((1-t)\theta)\mathbf{n_0} + \sin(t\theta)\mathbf{n_1}}{\sin(\theta)} \tag{5}$$

where $\mathbf{n_0}, \mathbf{n_1}$ is the initial and target normal respectively, θ is the angle between $\mathbf{n_0}, \mathbf{n_1}$. This approach preserves unit length normals but again the direction of the normal is far from being perpendicular to the intermediate triangles.

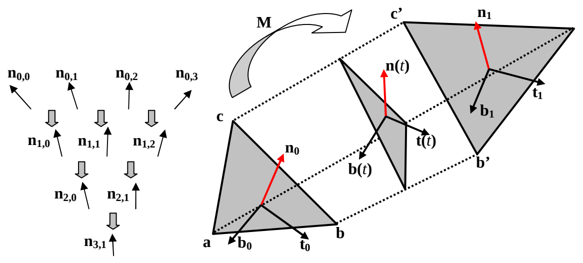


Figure 3: (a) a demonstration of deCasteljau algorithm for vectors, the gray thick arrow represents application of SLERP on two successive normals, (b) quaternion interpolation of face normal, transformation $\mathbf{M}$ transforms the triangle $\mathbf{a},\mathbf{b},\mathbf{c}$ to the triangle $\mathbf{a}',\mathbf{b}',\mathbf{c}'$ and $\mathbf{n_0}$ to $\mathbf{n_1}$. The frame $\mathbf{F_1} = (\mathbf{t_1},\mathbf{b_1},\mathbf{n_1})$ is computed by transforming $\mathbf{F_0} = (\mathbf{t_0},\mathbf{b_0},\mathbf{n_0})$ by the transformation $\mathbf{M}$.

2.4 Spherical deCasteljau

A slightly better idea is to compute several intermediate normals exactly (as in higher degree Lagrange interpolation) and interpolate these vectors on the surface of the unit sphere. In this case, a generalized deCasteljau algorithm for spherical interpolation can be used. The deCasteljau algorithm is well known for fast generation of Bezier curves. It is basically a recursive subdivision of the control polygon of the Bezier curve which converges very quickly to the curve. For example, let us have n vertices of a control polygon and we want to compute the point on the Bezier curve defined by the control polygon for the parameter value t_0. We subdivide each of the $n-1$ edges of the control polygon in the ration $t_0 : (1-t_0)$. It results in a new control polygon with $n-1$ vertices and $n-2$ edges. The same procedure is recursively repeated until only one edge remains and by subdivision of this edge we end up with a point on the curve. The generalization of deCasteljau algorithm for fast vector interpolation means that we replace line segments with shortest great circle arcs, i.e., the line segment subdivision step is replaced by SLERP of consecutive intermediate normals.

It is demonstrated in figure 3(a) where we have an initial normal $\mathbf{n_{0,0}}$, a final normal $\mathbf{n_{0,3}}$, and two intermediate normals $\mathbf{n_{0,1}}$ and $\mathbf{n_{0,2}}$, (e.g., in the time $t = 0.33$ and $t = 0.66$). When we want to compute normal $\mathbf{n_f}$ at time t we compute normals $\mathbf{n_{1,0}}$, $\mathbf{n_{1,1}}$, $\mathbf{n_{1,2}}$ by applying SLERP on pairs of successive normals, i.e., $\mathbf{n_{1,0}} = SLERP(\mathbf{n_{0,0}},\mathbf{n_{0,1}},t), \mathbf{n_{1,1}} = SLERP(\mathbf{n_{0,1}},\mathbf{n_{0,2}},t), \mathbf{n_{1,2}} = SLERP(\mathbf{n_{0,2}},\mathbf{n_{0,3}},t)$. This process is repeated until one single normal $\mathbf{n_{3,1}}$ is obtained.

2.5 Quaternion SLERP

In this section we will present our idea which is based on the use of quaternions [Shankel 2000] in order to interpolate the face normals. First let us recall two important identities, which we will use in the following description:

Definition 1: Rotation in 3D around an axis $\mathbf{a}$ by an angle ϕ is represented by a 3x3 rotation matrix $\mathbf{R}$ or by a quaternion $\mathbf{q}$ [Shoemake 1985], [Eberly 2004]. Both representations are equivalent.

Definition 2: The matrix $\mathbf{R}$ of a rotation transformation is orthogonal and its columns (and rows) are having unit length, thus the matrix of the rotation forms an orthonormal frame.

The central idea is to set an orthogonal frame $\mathbf{F_0} = (\mathbf{n_0},\mathbf{t_0},\mathbf{b_0})$ for the initial face and set an orthogonal frame $\mathbf{F_1} = (\mathbf{n_1},\mathbf{t_1},\mathbf{b_1})$ for the final face. To set a frame $\mathbf{F} = (\mathbf{n},\mathbf{t},\mathbf{b})$ means to associate one vector of the frame with the normal of the face $\mathbf{n}$, choose a tangent $\mathbf{t}$ which lies in the plane of the face (e.g., an edge of the triangle) and compute the binormal $\mathbf{b}$ by taking the cross product of $\mathbf{n}$ and $\mathbf{t}$, i.e., $\mathbf{b} = \mathbf{n} \times \mathbf{t}$. By organizing the column vectors $\mathbf{t},\mathbf{b},\mathbf{n}$ into a 3×3 matrix we obtain a rotation matrix $\mathbf{R}$ which can be converted into a quaternion representation $\mathbf{q}$ [Eberly 2004]. In this way we obtain quaternions $\mathbf{q_0}$ and $\mathbf{q_1}$, representing the initial and the final orientation of the face, respectively. To obtain intermediate normals $\mathbf{n}(t)$, quaternions are interpolated using quaternion SLERP (generalization of vector SLERP for quaternions in 4D), i.e.,

$$\mathbf{q}(t) = SLERP(\mathbf{q_0},\mathbf{q_1},t) \tag{6}$$

and intermediate quaternion $\mathbf{q}(t)$ can be converted back to the orthogonal matrix $\mathbf{R}(t)$ and the normal is extracted from the last column of $\mathbf{R}(t)$. It is demonstrated in figure 3(b) where the triangle $\mathbf{a},\mathbf{b},\mathbf{c}$ with the frame $\mathbf{F_0}$ is transformed in the triangle $\mathbf{a}',\mathbf{b}',\mathbf{c}'$ with the frame $\mathbf{F_1}$. The intermediate quaternion $\mathbf{q}(t)$ is converted to the frame $(\mathbf{t}(t),\mathbf{b}(t),\mathbf{n}(t))$ and the intermediate normal $\mathbf{n}(t)$ is extracted. The question is how to set the frames $\mathbf{F_0}$ and $\mathbf{F_1}$. We use following scheme. First we compute a transformation matrix $\mathbf{T}$ which transforms vertices of the initial triangle to the vertices of the final triangle, moreover we want that the transformation $\mathbf{T}$ transforms also the initial normal to the final normal. All condition can be expressed by a matrix equation, i.e.:

$$\mathbf{M}[\mathbf{a_0 b_0 c_0 n_0}] = [\mathbf{a_1 b_1 c_1 n_1}] \tag{7}$$

where $\mathbf{a_0}$, $\mathbf{b_0}$, etc. are column vectors, e.g., $\mathbf{a_0} = [\mathbf{a_{0_x}}, \mathbf{a_{0_y}}, \mathbf{a_{0_z}}, \mathbf{1.0}]^{\mathbf{T}}$. The matrix $\mathbf{M}$ can be computed as:

$$\mathbf{M} = [\mathbf{a_1 b_1 c_1 n_1}][\mathbf{a_0 b_0 c_0 n_0}]^{-\mathbf{1}} \tag{8}$$

Then we set an arbitrary frame $\mathbf{F_0}$ for the initial face. For the frame $\mathbf{F_1}$ we have to choose $\mathbf{t_1}$ and $\mathbf{b_1}$ since $\mathbf{n_1}$ is given. We compute $\mathbf{b_1}$ by transforming $\mathbf{b_0}$ by the matrix $\mathbf{M}$, i.e., $\mathbf{b_1} = \mathbf{Mb_0}$, $\mathbf{t_1}$ is then computed as cross product of $\mathbf{n_1}$ and $\mathbf{b_1}$, i.e., $\mathbf{t_1} = \mathbf{n_1} \times \mathbf{b_1}$.

3 Comparison and Discussion

We compared the approaches described in section 2 from two points of view. The first aspect is the quality of the interpolation and the second aspect is the time consumption. The quality of the interpolation is measured as follows. The time interval (usually $[0,1]$) is sampled and in each sample i an angle a_i between the exact normal (computed by the cross product) and interpolated normal (computed by some interpolation approach described in section 2) is computed. Angles between the exact normals and interpolated normals in individual samples are summed and the resulting number q represents the quality of the interpolation scheme, i.e. $q = \sum a_i, i = 1...n$ where n is the number of samples.

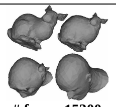
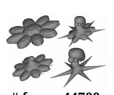
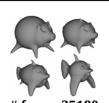
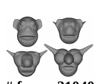
Normal interpolation approach	 **# faces: 15300**	 **# faces: 44700**	 **# faces: 35180**	 **# faces: 21040**
Linear	18916	55756	44078	20061
Quadratic	4820	15792	9681	5425
Cubic	1906	6444	3640	1572
t-variant cross product	0	0	0	0
Vector SLERP	19427	57999	44972	20861
Spherical deCasteljau	7039	21772	16068	7647
Quaternion SLERP	17766	53492	42831	19202

Table 1: Comparison of different normal interpolation approaches.

We tested different interpolation approaches on a morphing animation, where a mesh composed of triangles deforms from one shape to another shape so that the trajectories of individual vertices are linear. The numbers in table 1 represents a quality of interpolation for the whole mesh, it is computed as a sum of quality of interpolation of individual triangles of the mesh, i.e., $Q = \sum q_j, j = 0...m$

where m is the number of triangles of the mesh and q_j is the quality of interpolation of j-th triangle. The numbers in table 1 must always be viewed with respect to the number of triangles and the number of samples. Rather then absolute values it is more important to compare ratios between different methods, e.g., it can be seen that quadratic interpolation is almost 4-times better than simple linear interpolation. The second, third and fourth row shows the results of normal interpolation described in Section 2.2. The row t-variant cross product shows the result of the approach from the Section 2.1. The row Vector SLERP shows results of method described in the Section 2.3. Results of vector interpolation using generalized deCasteljau algorithm (Section 2.4) are shown in the Spherical deCasteljau row. The last row shows the quality of normal interpolation using quaternions.

From table 1 it is clear that the best approach from the quality point of view is the t-variant cross product approach which gives exact normals, i.e., it is not really an interpolation approach. The other approaches (except the quaternion approach) handles normal vectors as usual vectors, i.e., it is not respected that normals vector are perpendicular to some surface and in fact, these approaches can be used for the interpolation of any vectors. The second worst result is obtained by linear interpolation. Better results can be achieved by quadratic or cubic interpolation (up to 90% improvement). The worst result was achieved by SLERP of normal vectors. The quality of interpolation by spherical deCestaljau approach depends on how many intermediate normals we use. In this case we used initial and final normal and two additional intermediate normals. The quaternion SLERP approach has slightly better results (on average about 5%) than simple linear interpolation.

The problem of Quaternion SLERP approach is that there are an infinite number of frames configurations that represent the face orientation. First, we generated a pair of random frames $\mathbf{F_0}$ (for the initial face), $\mathbf{F_1}$ (for the final face) and we tested the quality of interpolation. A random frame is given by a normal $\mathbf{n}$ of the face, an arbitrary vector $\mathbf{t}$ lying in the plane of the face and a binormal vector $\mathbf{b}$ computed as $\mathbf{b} = \mathbf{n} \times \mathbf{t}$. Different configurations of random frames lead to different quality of interpolation. We did not succeed in devising a deterministic algorithm for computation of best pair of frames configuration (from the interpolation quality point of view). Our frame computation based on the transformation matrix in equation (8) which maps the initial face to the final face yields good results, however some specific frame configurations led to a better interpolation quality.

Next we will compare various approaches from the time consumption point of view. We will not present exact timing since it is dependent on how various elementary operations (SLERP, cross-product, polynomial evaluation, etc.) are implemented. Some of them can be implemented in hardware so that their execution can be very fast. We will express the time consumption in terms of elementary operations, so that one must always decide which approach is the most suitable according to the actual application platform.

Elementary operations used in table 2 are polynomial evaluation, SLERP and quaternion to matrix conversion. Polynomial evaluation is used in the Lagrange interpolation approach and in the t-variant cross product approach. Polynomials can be evaluated by the Horner scheme which saves some multiplications in comparison with usual evaluation of polynomial in monomial form. SLERP is used in the Vector SLERP approach and in the deCasteljau approach. SLERP requires evaluation of trigonometric functions which are computationally expensive, but it can be speed up by an incremental approach which was described Barrera et al in [Barrera 2004].

Quaternion to matrix conversion is used in the Quaternion SLERP approach. Note that in our case we need to extract only one column from the matrix, i.e., the normal. It is also important to decide whether we need random access to the deformation or just sequential access. Random access means that we can jump from one time instant to another. Sequential access means that the mesh deformation is sequentially played; in this case incremental methods (using temporal coherence) for computing SLERP [Barrera 2004] or polynomial evaluation [Hast 2003] can be used. For example, a degree two polynomials can be evaluated by an incremental method using only two additions.

Method	**Requires normalization**	**Computation**
Linear interpolation	yes	3 linear interpolation, Eq. (4)
Quadratic, cubic interpolation	yes	3 evaluation of degree two (quadratic interp.) or degree three (cubic interp.) polynomials
t-variant cross product	yes (Eq. 1) no (Eq. 2)	3 evaluation of degree two polynomials (Eq. 2) 3 evaluation of rational polynomial (Eq. 3)
Vector SLERP	no	1 SLERP
Spherical deCasteljau	no	n(n-1)/2 SLERPs, where n is the number in precomputed normals
Quaternion SLERP	no	1 SLERP, quaternion to matrix conversion

Table 2: Comparison of various normal interpolation approaches from time consumption point of view in terms of elementary operations.

4 Conclusions and Future Work

In this paper we showed five approaches on how to compute normals of triangular faces under linear deformation. It is clear that the Linear interpolation approach always will be faster than any other approach but it will also have very bad results when the deformation is dramatic. The higher degree Lagrange interpolation is better but still it is just an interpolation of vectors and in some cases it does not capture the face deformation well.

The t-variant cross product approach gives exact face normals; the lengths of normals are proportional to the size of face so it can be directly used for vertex normal computation weighted by the area [Parus 2006]. If unit length normals are required, both Lagrange interpolation and t-variant cross product approaches require additional normalization.

Approaches based on SLERP do not require vector normalization since the vectors are interpolated on the surface of the unit sphere. The problem of Spherical deCasteljau approach is that it interpolates the initial and final normal but it just approximates the intermediate normals (in the same way how Bezier curve approximates the control polygon). So it is not useful when we want interpolate exactly some vector different then the initial and the final.

The new quaternion interpolation approach is different from previous approaches (except t-variant cross product approach) because it interpolates frames instead of vectors. Interpolation of the frame may capture better the true triangle motion and thus it can produce more accurate normals then the simple Vector SLERP approach. Like the linear interpolation approach and the Vector SLERP approach it has the advantage that no intermediate normals are necessary to compute exactly in order to set up the interpolation for the animation.

In the future work we would like to study the Quaternion interpolation approach. It has promising results because by experiments we found out that some frame configurations lead to a better quality of interpolation than others, so we would like to find such frame configurations which lead to the best possible quality of the normal interpolation.

References

T. Barrera, A. Hast, E. Bengtsson 2005. *Incremental Spherical Linear Interpolation* SIGRAD 2004, pp. 7-10.

D. EBERLY 2004. *Game Physics* Morgan Kaufman, 2004

A. GLASSNER 1999. *Situation Normal* Andrew Glassner's Notebook- Recreational Computer Graphics, Morgan Kaufmann Publishers, pp. 87-97.

J. GOMES 1999. *Warping and Morphing of Graphical Objects* Morgan Kaufman. San Francisco, California. 1999.

H. GOURAUD 1971. *Continuous Shading of Curved Surfaces* IEEE Transactions on Computers, Vol. 20, No. 6, 1971.

A. HAST, T. BARRERA, E. BENGTSSON 2003. *Improved Shading Performance by avoiding Vector Normalization,* WSCG'01, Short Paper,2001, pp. 1-8.

S. JIN 2005. *A Comparison of Algorithms for Vertex Normal Computation* Communications of the ACM, Vol. 18, No 6, 1975

Z. KARNI 2004. *Compression of soft-body animation sequences* Computers and Graphics, 28: 25-34. 2004.

J. PARUS, I. KOLINGEROVÁ 2006. *Normal Evaluation for Soft-body Deforming Meshes* SimVis2006, pp. 157-168, 2006.

B. T. PHONG 1975. *Illumination for Computer Generated Pictures* The Visual Computer, Springer-Verlag GmbH, Issue: Vol. 21, No 1-2, pp. 71-82, Feb. 2005.

J. SHANKEL 2000. *Interpolating Quaternions* Game Programming Gems. Edited by M. DeLoura. Charles River Media, pp. 205-213

K. SHOEMAKE 1985. *Animating rotation with quaternion curves* ACM SIGGRAPH, pp. 245-254.