Tetrisation of triangular meshes and its application in shape blending

Kaji, Shizuo
Department of Mathematical Sciences, Faculty of Science, Yamaguchi University : Lecturer | JST CREST

出版情報：MI lecture note series. 64, pp.33-42, 2015-09-18. 九州大学マス・フォア・インダストリ研究所
バージョン：
権利関係：

MATHEMATICAL PROGRESS IN EXPRESSIVE IMAGE SYNTHESIS 2015
Sep 25-27, 2015, Nishijin Plaza Kyushu University, Fukuoka

Tetrisation of triangular meshes and its application in shape blending

Shizuo KAJI*
Yamaguchi University / JST CREST

Abstract We introduce a method to associate a tetrahedral structure to a triangular mesh in 3D space, which can then be used in shape deformation applications. Our method allows one to keep track of global geometry such as curvature while working locally on tetrahedra. We apply this method to give a shape blending algorithm based on the as-rigid-as-possible deformation scheme. An implementation and its MIT licensed source code is available online.

Keywords: shape blending, tetrahedral mesh, as-rigid-as-possible deformation

1 Introduction

In mesh editing applications, it is sometimes required to have a tetrahedral structure rather than a triangular mesh. For example, in the seminal paper [1] they used tetrahedral volume meshes to produce interpolation of shapes. However, in most computer graphic systems it is common to represent shapes by surface meshes. To convert a surface mesh to a volume mesh is a non-trivial task (see, for example, [15]) and the resulting volume mesh tends to have many extra internal vertices, which make applications inefficient.

Instead of considering volume meshes, one can "fatten" surface meshes. A common practice is to associate a tetrahedral structure to a triangular surface mesh by adding the normal vector for every triangle (see, for example, [17]). This simple trick has been widely used, and also has a nice theoretical interpretation in discrete differential geometry. However, it does not capture important geometric features of the mesh. For example, the relation between adjacent triangles is neglected.

The purpose of this paper is to introduce two more constructions to associate a tetrahedral structure to a triangular mesh, which we call *tetrisation*. Our method encodes inter-triangular properties such as the angle between adjacent triangles.

As an application, we give a shape blending algorithm based on [1] (Fig. 1). Given an arbitrary number of isomorphic surfaces, our algorithm produces inter/extrapolation of the shapes according to the weights given by the user. Roughly speaking, we define a "linear combination" of shapes

$$w_1 Q_1 + w_2 Q_2 + \cdots + w_m Q_m,$$

where $w_i \in \mathbb{R}$ are weights and Q_i are shapes. Note that our algorithm is highly non-linear although we described the procedure as taking the linear combination of shapes. We implemented the algorithm as the Autodesk Maya plugin. Its MIT licensed source code is available at [8].

*skaji@yamaguchi-u.ac.jp

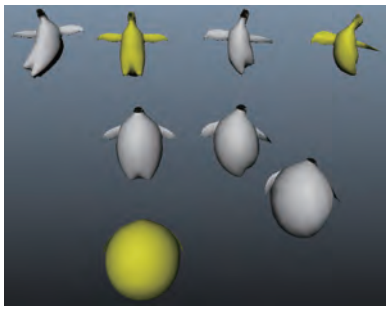

Figure 1: Three shapes (yellow) are blended to produce variations (white). Not only interpolation but also extrapolation (with weights > 1 or < 0) is possible. The top-left shape is obtained by extrapolating the two yellow shapes in the top row.

2 Notation

First, we list the notation used in this paper. We assume all the tranformations are represented by real matrices acting on real column vectors by the multiplication from the left.

- $\mathrm{SO}(3)$: the group of 3D rotations. Its element is a 3×3 special orthogonal matrix.
- $\mathrm{Sym}^+(3)$: the set of 3D shears. Its element is a 3×3 positive definite symmetric matrix.
- $\mathrm{GL}(3)$: the group of 3D linear transformations (compositions of rotation, shear, and reflection). Its element is a 3×3 regular matrix.
- $\mathrm{Aff}(3)$: the group of the 3D affine transformations (compositions of rotation, shear, reflection, and translation). Its element is a 4×4 homogeneous matrix.
- $\mathrm{GL}^+(3), \mathrm{Aff}^+(3)$: the subgroups of the reflection free (positive determinant) elements in the corresponding groups.
- $\hat{A} \in \mathrm{GL}(3)$: the linear part ($3 \times 3$ upper-left corner) of $A \in \mathrm{Aff}(3)$.
- A^t: the transpose of a matrix A.
- $|A|_F^2 = \mathrm{tr}(A^t A)$: the squared Frobenius norm of a matrix A.
- $\#U$: the cardinality of a set U.

3 As-rigid-as-possible deformation framework

First, we recall the framework of the *as-rigid-as-possible deformation technique* (ARAP, for short) since it is the main application and motivation of our work. The ARAP was first introduced in [1] for interpolating two shapes and has provided one of the fundamental frameworks for shape deformation (see, for example, [3, 7, 9, 16, 17, 18]). The idea is to measure the global error of a deformed shape as the sum of the local error over the tetrahedra. The local error is, in turn, defined in terms of the matrix which maps an tetrahedron of the original shape to the corresponding one of the deformed shape. We will discuss it precisely in the following.

Definition 1. A tetrahedral structure is a pair $(V, \mathcal{T})$, where the vertex set V consists of three dimensional vectors and the set of tetrahedra $\mathcal{T} = \{T_i \mid 1 \leq i \leq n\}$ consists of ordered tuples of four distinct vertices $T_i = (v_{i_1}, v_{i_2}, v_{i_3}, v_{i_4})$. Each vertex must be contained in at least one tetrahedron.

A tetrahedral structure is said to be non-degenerate when the vertices of each tetrahedron are not co-planar.

We emphasise that overlap between tetrahedra is allowed, and for this reason, we use the terminology "tetrahedral structure" rather than tetrahedral mesh.

Let us assume that we are given an original shape with a tetrahedral structure $Q = (V, \mathcal{T})$. By a deformation Q' of Q, we mean a tetrahedral structure with a vertex set V' of the same cardinality as V and the same set of tetrahedra $\mathcal{T}$. That is, only the vertex positions are modified.

We can pack the information of V and $\mathcal{T}$ into a set of 4×4-matrices:

$$\left\{ P_i := \begin{pmatrix} v_{i_1}(x) & v_{i_2}(x) & v_{i_3}(x) & v_{i_4}(x) \\ v_{i_1}(y) & v_{i_2}(y) & v_{i_3}(y) & v_{i_4}(y) \\ v_{i_1}(z) & v_{i_2}(z) & v_{i_3}(z) & v_{i_4}(z) \\ 1 & 1 & 1 & 1 \end{pmatrix} \middle| 1 \leq i \leq n \right\}, \tag{1}$$

where $(v_{ij}(x), v_{ij}(y), v_{ij}(z))^t \in \mathbb{R}^3$ is the vector representing the position of the vertex $v_{ij} \in V$.

Assume that the original tetrahedral structure Q is non-degenerate so that all the P_i are regular. Then, we can define a series of affine transformations

$$A_i := P_i' P_i^{-1} \ (1 \leq i \leq n) \tag{2}$$

which maps the vertices V of the original shape Q to the ones V' in the deformed shape Q'.

Now, suppose the local transformations $C_i \in \mathrm{GL}^+(3)$ $(1 \leq i \leq n)$ are prescribed. It depends on applications how C_i are given. They are usually calculated from the user's input (see §5 or [1, 9]). We compute the deformed shape Q' such that V' minimises an *error function* with respect to C_i. We consider two different error functions. In [1], they introduced

$$E_T(V', \{C_i\}) = \sum_{i=1}^{n} |\hat{A}_i - C_i|_F^2, \tag{3}$$

where $\hat{A}_i$ is thought of as a linear function of V' by (2). Many of ARAP based shape deformation applications including [2, 17, 18, 19] use this function. Note that this is translation invariant but not rotation invariant. Rotation invariance is sometimes preferable in shape deformation (see, for example, [13, 16] and Fig. 8). We propose a rotation and translation invariant error function defined by

$$E_S(V', \{C_i\}) = \sum_{i=1}^{n} |S(\hat{A}_i) - S(C_i)|_F^2, \tag{4}$$

where $S(X)$ for $X \in \mathrm{GL}(3)$ is the shear factor of the polar decomposition of X (see [14]). Intuitively, this error function measures how much each tetrahedron is distorted. Despite the simplicity and its invariance property, E_S has not been considered in the literature as far as the author is aware. We believe this error function gives a good alternative to E_T in some applications (see Fig. 8).

Remark 2. We can assign a weight $W_i \in \mathbb{R}$ to each tetrahedron T_i to specify its contribution to the error function. It is done simply by replacing the summation $\sum_{i=1}^n$ with the weighted one $\sum_{i=1}^n W_i$ in the definitions of the error functions. For notational simplicity, we omit them in this paper.

The deformed shape is computed as the minimiser of the error function. Note that there is some indeterminacy of the minimiser coming from the invariance. For example, any translation of a minimiser is also a minimiser. To obtain a unique minimiser, one can impose additional constraints; for E_T fixing the position of the barycentre and for E_S fixing the position of the barycentre and the orientation of some tetrahedra.

Computing the minimiser for E_T is reduced to solving a sparse linear system (see [1, 18]). For E_S, the computation is not linear. An iterative way similar to [16] is given as follows:

1. Compute the minimiser of $E_T(V', \{C_i\})$ and set $\hat{A}_i$.
2. Compute the polar decomposition $\hat{A}_i = R_i S_i$.
3. Compute the minimiser of $E_T(V', \{R_i S(C_i)\})$ to update $\{\hat{A}_i\}$.
4. Repeat (2) and (3) until $\{\hat{A}_i\}$ converge.

4 Tetrisation

In computer graphics systems, shapes are usually represented by surface meshes. Therefore, we cannot directly apply the ARAP technique, which requires a tetrahedral structure. Here, we consider a method to obtain a tetrahedral structure from a given triangular mesh.

For a triangular mesh, we denote an element of the vertex set V by a three dimensional vector and an element of the set of (face) triangles $\mathcal{F}$ by an ordered tuple of three vertices (v_1, v_2, v_3). For $(v_1, v_2, v_3) \in \mathcal{F}$, we call the ordered tuples v_1v_2, v_2v_3, v_3v_1 the *oriented edges*. A triangular mesh is said to be *non-degenerate* when the vertices of each triangle are not co-linear.

We associate to a triangular mesh a tetrahedral structure with which one can work in the ARAP framework.

Definition 3. Given a non-degenerate triangular mesh $(V, \mathcal{F})$. A *tetrisation* of $(V, \mathcal{F})$ is a tetrahedral structure which consists of the vertex set $\bar{V}$ and the set of tetrahedra $\mathcal{T}$. We require $\mathcal{T}$ to satisfy the following conditions:

1. $V \subset \bar{V}$. That is, $\bar{V}$ is obtained by adding *ghost vertices* to V.
2. Each triangle in $\mathcal{F}$ has to be contained in at least one tetrahedron in $\mathcal{T}$.
3. Each tetrahedron is non-degenerate, that is, the four vertices are not co-planar.

These conditions are exactly what are required in the ARAP framework.

We give three methods to produce tetrisation in the following. First, recall that the unit normal vector $n(T)$ of a triangle $T = (v_1, v_2, v_3)$ is computed by $\dfrac{(v_2 - v_1) \times (v_3 - v_1)}{|(v_2 - v_1) \times (v_3 - v_1)|}$, where the denominator $|(v_2 - v_1) \times (v_3 - v_1)|$ is twice the area $2\mathrm{Area}(T)$ of T.

4.1 Face-normal tetrisation

We begin with a simplest method which has been commonly used in various applications. For each triangle $T = (v_1, v_2, v_3)$ in $\mathcal{F}$, add the ghost vertex

$$v_0 = \frac{(v_1 + v_2 + v_3)}{3} + \frac{(v_2 - v_1) \times (v_3 - v_1)}{\sqrt{|(v_2 - v_1) \times (v_3 - v_1)|}}$$

and form a tetrahedron (v_0, v_1, v_2, v_3). The resulting tetrahedral structure has $\#\mathcal{T} = \#\mathcal{F}$ and $\#\bar{V} = \#V + \#\mathcal{T}$.

A problem with this tetrisation when applied to the ARAP framework is that this does not capture the relation between adjacent triangles. For example, consider two triangles sharing an edge as in Fig. 2. The rotation invariant error function E_S with $C_1 = C_2 = Id$ will be minimised with any angle between the two triangles. In other words, folds do not cause any penalty in the error function.

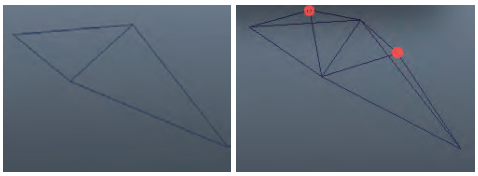

Figure 2: Left: the original surface, Right: its face-normal tetrisation. Ghost vertices are marked with a red circle.

4.2 Edge-normal tetrisation

We assume each oriented edge appears only once among all the triangles. In other words, an unoriented edge should be contained at most two triangles with the opposite orientations. Also, we assume all the triangles have at least one shared edge, that is, there is no "lone" triangle. (We can remove this assumption by adding ghost vertices not only for shared edges but for all edges. However, this is inefficient and makes no sense.)

For each shared edge v_1v_2, denote by $T_1 = (v_1, v_2, v_3)$ and $T_2 = (v_1, v_4, v_2)$ the two triangles adjacent to it. Add a ghost vertex

$$v_0 = \frac{v_1 + v_2}{2} + |v_1 - v_2| \frac{n(T_1) + n(T_2)}{|n(T_1) + n(T_2)|}$$

and form two tetrahedra (v_0, v_1, v_2, v_3) and (v_0, v_1, v_4, v_2). The resulting tetrahedral structure has $\#\mathcal{T} = 2 \cdot \#(\text{shared edges})$ and $\#\bar{V} = \#V + \#(\text{shared edges})$.

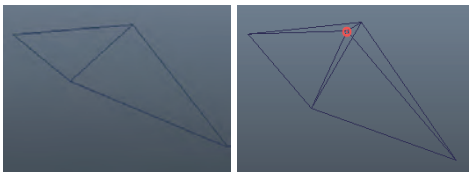

Figure 3: Left: the original surface, Right: its edge-normal tetrisation. Ghost vertex is marked with a red circle.

The idea of this tetrisation is to encode the angle between adjacent triangles, which is neglected by the face-normal tetrisation.

4.3 Vertex-normal tetrisation

We assume that every vertex has a neighbourhood homeomorphic to the plane or the half plane. In other words, the mesh is a manifold (with boundary). Also, we assume all the triangles have at least one shared vertex. (Again, we can remove this assumption as in the previous subsection.)

For each shared vertex v, denote by $T_1, T_2, \ldots, T_k$ the adjacent triangles. Add a ghost vertex

$$v_0 = v + \sqrt{\sum_{i=1}^{k} \text{Area}(T_i)} \frac{n(T_1) + \cdots + n(T_k)}{|n(T_1) + \cdots + n(T_k)|}$$

and form k tetrahedra by adding v_0 to the triangles T_i $(1 \leq i \leq k)$. The resulting tetrahedral structure has $\#\mathcal{T} = 3\#\mathcal{F} - \#(\text{unshared vertices})$ and $\#\bar{V} = \#V + \#(\text{shared triangles})$.

An advantage of this method is that it extends straightforwardly to general polyhedral meshes.

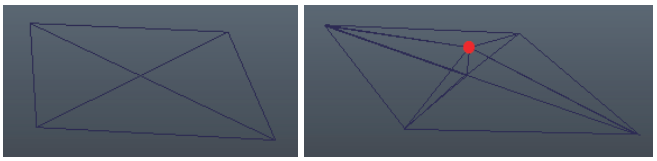

Figure 4: Left: the original surface, Right: its vertex-normal tetrisation. Ghost vertex is marked with red circle. We omitted ghost vertices on the boundary vertices for simplicity.

The idea of this tetrisation is to encode the angle around internal vertices, which is neglected by the face-normal tetrisation.

5 Application: Shape blending

Finally, we discuss an application of our method to shape blending. Roughly speaking, shape blending produces a "linear combination" of shapes

$$w_1 Q_1 + w_2 Q_2 + \cdots + w_m Q_m,$$

where $w_i \in \mathbb{R}$ are weights and Q_i are shapes. In particular, when the number of shapes is two, $w_1 Q_1 + (1 - w_1) Q_2$ for $0 \leq w_1 \leq 1$ gives a morphing between them. A basic method is to simply take the linear combination of the coordinates of the vertices. This algorithm is very fast and widely used to produce variations of shapes, in particular, facial expressions (see, for example, [12]). However, since the geometry of shapes is disregarded, it does not always produce plausible outputs (Fig. 5).

Figure 5: Interpolation between yellow shapes. Left: linear blend shape, Right: our method

In [18] they give an algorithm based on [1] to blend/morph two or more shapes while preserving the geometry of shapes. A 2D version is also discussed in [2]. Here, we propose a similar algorithm based on [1]. The main differences between the method which we present here and theirs are

1. [18] considers only the face-normal tetrisation
2. [18] uses the error function E_T, which is not rotation invariant
3. [18] uses the polar decomposition ([14] and [5]) to interpolate affine transformations, while we use the Cartan decomposition based parametrisation given in [10].

We will explain the difference and see its visual consequences later.

The problem setting is as follows. We are given a shape represented as a non-degenerate triangular mesh $(V_0, \mathcal{F})$ and m its deformations V_j $(1 \leq j \leq m)$. The system computes the deformation $V(w_1, \ldots, w_m)$ by blending the given shapes according to the user specified weights $\{w_j \in \mathbb{R} \mid 1 \leq j \leq m\}$. We insist that it interpolates the given shapes, i.e., $V(0, \ldots, 0) = V_0$, and $V(w_1, \ldots, w_m) = V_k$ when $w_j = \begin{cases} 1 & (j = k) \\ 0 & (j \neq k) \end{cases}$. Notice we allow negative weights so that the system can extrapolate not only interpolate.

We describe a general scheme based on the ARAP. The first step is to assign a tetrahedral structure $(\bar{V}, \mathcal{T})$ to the triangular mesh $(V_0, \mathcal{F})$ by tetrisation. Then, by (2) we obtain a set of affine transformations

$$A_{ji} := P_{ji} P_{0i}^{-1} \in \mathrm{Aff}^+(3) \ (1 \leq j \leq m, 1 \leq i \leq n),$$

where P_{ji} are as in (1) created from V_j. Notice that by virtue of the non-degeneracy condition of tetrisation, each A_{ji} has the positive determinant.

The second step is to compute the "blended local transformations" $C_i \in \mathrm{GL}^+(3)$ by blending A_{ji} according to the weights w_j. Intuitively, C_i stipulates the local transformation for the tetrahedron T_i. To define C_i, we can use any interpolation function Blend for $\mathrm{GL}^+(3)$ which satisfies the obvious requirement for interpolation and set

$$C_i := \mathrm{Blend}(w_1, \ldots, w_m, \hat{A}_{1i}, \hat{A}_{2i}, \ldots, \hat{A}_{mi}).$$

We give two such interpolation functions later.

The third and final step is to determine the positions of the vertices $V(w_1, \ldots, w_m)$ as the minimiser of any error function in §3. Since C_i are generally not compatible on shared faces, we need to "patch" them by finding the global piecewise linear transformation which minimises the error function.

Now, we recall two interpolation functions for $\mathrm{GL}^+(3)$. By the polar decomposition (see, for example, [14]), we obtain

$$\hat{A}_{ki} = R_{ki} S_{ki}$$

where $R_{ki} \in \mathrm{SO}(3)$ is the rotation and $S_{ki} \in \mathrm{Sym}^+(3)$ is the shear. [18] suggests

$$\mathrm{Blend}_P(w_1, \ldots, w_m, \hat{A}_{1i}, \hat{A}_{2i}, \ldots, \hat{A}_{mi}) = \sum_{k=1}^{m} \exp(w_k \log(R_{ki})) \left(\sum_{k=1}^{m} w_k S_{ki} + \left(1 - \sum_{k=1}^{m} w_k\right) I \right),$$

where $\log$ is the principal matrix logarithm and I is the identity matrix[1]. On the other hand, we suggest to use [10]

$$\mathrm{Blend}_C(w_1, \ldots, w_m, \hat{A}_{1i}, \hat{A}_{2i}, \ldots, \hat{A}_{mi}) = \sum_{k=1}^{m} \exp(w_k \log^c(R_{ki})) \sum_{k=1}^{m} \exp(w_k \log(S_{ki})),$$

where $\log^c$ is the "continuous" logarithm such that it chooses the nearest branch of logarithm to the adjacent tetrahedra when i varies (see [10] for details). The indeterminacy of $\log$ for $\mathrm{SO}(3)$ is in the rotation angle and $\log^c$ chooses the angle continuously for adjacent tetrahedra. Note that [10] provides a direct and fast formula for Blend_C which does not require the polar decomposition.

They look similar but there are two significant differences; logarithm for $\mathrm{SO}(3)$ and blending for the shear part. The value of Blend_P can fall out of $\mathrm{GL}^+(3)$ due to the linear blending of the shear part, which causes distortion in the output (Fig. 6). The use of the continuous logarithm enables

[1] The term involving I is for normalisation and it enforces $\mathrm{Blend}_P(0, \ldots, 0, \hat{A}_{1i}, \hat{A}_{2i}, \ldots, \hat{A}_{mi}) = I$.

the system to produce a smoother morph among shapes which performs large rotation in between (Fig. 7). Note in [1] which discusses morphing of two shapes, they suggest to use the quaternions and SLERP to interpolate the rotation part and the linear interpolation for the shear part. (With three or more shapes, one can use the linear blending of the quaternions for the rotation part as in [11].) However, this method shows similar deficiency as Blend_P.

Figure 6: Interpolation/extrapolation of yellow shapes. Left: with Blend_P function in [18], the extrapolated shape on the left is degenerate. Right: with our Blend_C function, the extrapolated shape is non-degenerate.

Figure 7: Interpolation of yellow shapes. Left: with Blend_P function in [18], some parts try to rotate inconsistently. Right: with our Blend_C function, local rotations are appropriately handled to produce a smooth interpolation

Fig. 8 visually compares different tetrisations in §4 and the error functions E_T and E_S in §3. We observe that E_S produces more natural results than E_T but much slower as we see in Table 1. With E_S, the face-normal tetrisation causes extra wrinkles compared to the edge-normal and the vertex-normal tetrisations. As far as we experimented, it depends on the character of shapes to be blended which tetrisation gives the best result. In general, with E_T the output is more or less similar regardless of the choice of tetrisation. With E_S, the vertex-nomal tetrisation seems to be a good choice.

Table 1 shows a timing comparison for different tetrisations and error functions. We blended two 3D models each with 26k triangles on a Macbook Air with 1.7Ghz Intel Core i7 and 8GB memory. Initialisation part involves the Cholesky decomposition of the space matrix necessary to solve the minimiser of the error functions. This is computed only once in the initialisation process. Note that the matrix is dependent on the tetrahedral structure but independent of the choice of the error function. Runtime part consists of finding the minimiser of the error functions and the computation of Blend functions.

	face E_T	edge E_T	vertex E_T	face E_S	edge E_S	vertex E_S
Initialisation (sec)	0.1976	0.3080	0.3556	0.1976	0.3080	0.3556
Runtime with Blend_P	45.45 fps	27.43 fps	30.86 fps	11.66 fps	4.132 fps	4.762 fps
Runtime with Blend_C	48.46 fps	29.31 fps	31.46 fps	12.19 fps	4.576 fps	5.037 fps

Table 1: Timing comparison

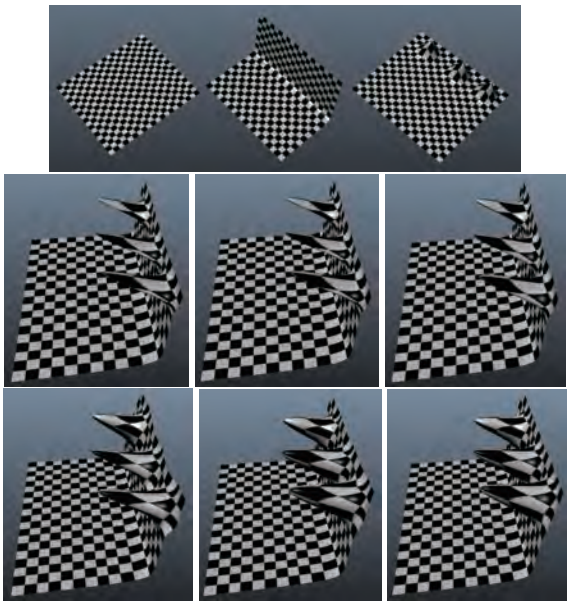

Figure 8: Top row from left to right: rest shape V_0 and its two deformations V_1 and V_2 to be blended with weights $w_1 = 1.0$ and $w_2 = 1.5$. Second row from left to right: results obtained by face-normal, edge-normal, and vertex-normal tetrisation with E_T, Third row: same as the second row but with E_S.

We implemented our algorithm as the Autodesk Maya plugin ([8]). In our system, the user can specify the weight for each shape by slider, or the ball controller which computes the weight by [4] from the configuration of the balls representing the shapes (Fig. 9).

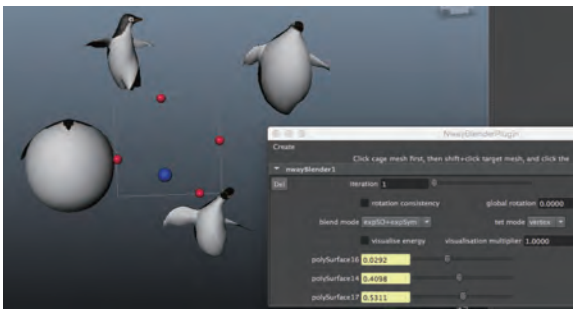

Figure 9: Our Maya plugin

Acknowledgement This work was partially supported by the Core Research for Evolutional Science and Technology (CREST) Program titled "Mathematics for Computer Graphics" of the Japan Science and Technology Agency (JST), by KAKENHI Grant-in-Aid for Young Scientists (B) 26800043, and by JSPS Postdoctoral Fellowships for Research Abroad.

References

[1] M. Alexa, D. Cohen-Or, and D. Levin, *As-Rigid-As-Possible Shape Interpolation*, Proc. ACM SIGGRAPH 2000, pp. 157–164 (2000)

[2] W. Baxter, P. Barla, and K. Anjyo, *N-way morphing for 2D animation*, Computer Animation and Virtual Worlds 20, 2-3, 79–87 (2009).

[3] M. Botsch and O. Sorkine, *On Linear Variational Surface Deformation Methods*, IEEE Transactions on Visualization and Computer Graphics 14(1), 213–230 (2008)

[4] M. S. Floater, *Mean value coordinates*, Computer Aided Geometric Design 20(1), 19–27 (2003)

[5] F. S. Grassia, *Practical parameterization of rotations using the exponential map*, J. Graph. Tools 3(3) , 29–48 (1998)

[6] N. Higham, *Computing the Polar Decomposition–With Applications*, SIAM J. Sci. and Stat. Comp. 7(4), 1160–1174 (1986)

[7] T. Igarashi, T. Moscovich, and J. F. Hughes, *As-rigid-as-possible shape manipulation*, ACM Transaction on Graphics 24(3), 1134–1141 (2005)

[8] S. Kaji, *An N-way morphing plugin for Autodesk Maya*, `https://github.com/shizuo-kaji/NWayBlenderMaya`

[9] S. Kaji and G. Liu, *Probe-type deformers*, Mathematical Progress in Expressive Image Synthesis II, Springer-Japan, 63–77 (2015)

[10] S. Kaji and H. Ochiai, *A concise parametrisation of affine transformation*, preprint, arXiv:1507.05290

[11] L. Kavan, S. Collins, J. Zara, and C. O'Sullivan, *Geometric Skinning with Approximate Dual Quaternion Blending*. ACM Transaction on Graphics 27(4), 105:1–105:23 (2008)

[12] J. P. Lewis and K. Anjyo, *Direct manipulation blendshapes*, IEEE Computer Graphics and Applications 30(4), 42–50 (2010)

[13] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or., *Linear rotation-invariant coordinates for meshes*, ACM Transaction on Graphics 24(3), 479–487 (2005)

[14] K. Shoemake and T. Duff, *Matrix animation and polar decomposition*, Proc. Graphics interface '92, Kellogg S. Booth and Alain Fournier (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 258–264 (1992)

[15] H. Si, *TetGen, a Delaunay-based quality tetrahedral mesh generator*, ACM Trans. Math. Softw. 41(2), 11:1–11:36 (2015)

[16] O. Sorkine and M. Alexa, *As-rigid-as-possible surface modeling*, Proc. Eurographics SGP '07, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 109–116 (2007)

[17] R. W. Sumner and J. Popović, *Deformation transfer for triangle meshes*, ACM Transactions on Graphics 23(3), 399–405 (2004)

[18] R. W. Sumner, M. Zwicker, C. Gotsman, and J. Popović, *Mesh-based inverse kinematics*, ACM Transaction on Graphics 24(3), 488–495 (2005)

[19] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H-Y Shum, *Mesh Editing with Poisson-based Gradient Field Manipulation*, ACM Transaction on Graphics 23(3), 644–651 (2004)