Mathematical Progress in Expressive Image Synthesis
Oct 21-23, 2013, Centennial Hall, Kyushu University School of Medicine, Fukuoka

A Lie Theoretic Parameterization of Affine Transformation

S. Kaji[1], S. Hirose[2], H. Ochiai[3], and K. Anjyo[4]
Yamaguchi University[1], OLM Digital, Inc.[24], Kyushu University[34], JST/CREST[1234]

Affine transformation (or geometric transformation) provides a mathematical foundation for shape manipulation and motion analysis in computer graphics. In particular, the set $\mathrm{Aff}^+(3)$ of *positive* (or, *reflection free*) affine transformations is important since it consists of rotation, shear, translation, and their compositions. The elements in $\mathrm{Aff}^+(3)$ are usually represented by 4×4-homogeneous matrices with algebraic operations such as addition, scalar product, and product. While the product corresponds to the composition of the transformations, geometric meaning of addition and scalar product are not clear. There are many situations where we want to have geometrically meaningful weighted sum (linear combination) of transformations, for example, for skinning [10], and for motion analysis and compression [1]. To mention a few parameterization developed previously: *Euler angle*, and *Quaternion* parameterizes the rotation. *Dual quaternion*, and *axis-angle presentation* parameterizes the rigid transformation (rotation and translation altogether). The above parameterizations are all partial; they deal only subsets of $\mathrm{Aff}^+(3)$, and cannot handle shear and scale. On the other hand, Alexa [1] introduced a Euclidean parameterization of $\mathrm{Aff}^+(3)$ using the *Lie correspondence*. The idea is that $\mathrm{Aff}^+(3)$ forms a *Lie group* and it corresponds to a linear space called *Lie algebra* through the matrix exponential and logarithm. However, this method yet fails to give a parameterization for the whole transformations; it is limited for transformations without negative eigenvalues. The limitation is due to mathematical nature of the Lie correspondence, which guarantees only *local* bijectivity.

Here we introduce a novel parameterization of $\mathrm{Aff}^+(3)$ based on Lie theory. Our general framework can also be found in [12], which includes more precise definitions of Lie group and Lie algebra. It has several advantages over previous ones;

- No limitation; it parameterizes the whole transformations.
- Smooth and having the same degree of freedom; ordinary variational techniques can be applied.
- With geometrically meaningful operations; for example, the sum of two rigid transformations is again rigid.
- Low computational cost; fast enough for real-time applications.

In the following sections, we will give the new parameterization method, a computation algorithm, and applications to shape deformation.

1. Parameterization of affine transformation

The elements of $\mathrm{Aff}^+(3)$ are represented by 4×4-homogeneous matrices whose linear parts have positive determinants:

$$\mathrm{Aff}^+(3) = \left\{ A = \begin{pmatrix} \hat{A} & d_A \\ 0 & 1 \end{pmatrix} \mid \det(\hat{A}) > 0, d_A \in \mathbb{R}^3 \right\}.$$

We follow the convention that column vectors are multiplied by matrices from the left.

Let $M_n(\mathbb{R})$ be the set of $n \times n$-matrices. We set the 12-dimensional parameter space

$$\mathfrak{se}(3) \times \mathfrak{sym}(3),$$

where

$$\mathfrak{se}(3) := \left\{ X = \begin{pmatrix} \hat{X} & l_X \\ 0 & 0 \end{pmatrix} \mid \hat{X} = -{}^t\hat{X} \in M_3(\mathbb{R}), l_X \in \mathbb{R}^3 \right\}$$

is the Lie algebra for the 3-dimensional rigid transformation group SE(3) and

$$\mathfrak{sym}(3) := \left\{ Y \mid Y = {}^tY \in M_3(\mathbb{R}) \right\}$$

is the set of the 3×3-symmetric matrices.

Now we define the parameterization map

$$(1) \qquad \begin{array}{rrcl} \phi: & \mathfrak{se}(3) \times \mathfrak{sym}(3) & \to & \mathrm{Aff}^+(3) \\ & X \times Y & \mapsto & \exp(X)\iota(\exp(Y)), \end{array}$$

where exp is the matrix exponential defined by

$$\exp(B) = \sum_{k=0}^{\infty} B^k/k!,$$

and $\iota : M_3(\mathbb{R}) \to M_4(\mathbb{R})$ is given by

$$\iota(B) = \begin{pmatrix} B & 0 \\ 0 & 1 \end{pmatrix}.$$

This gives a mathematically well-defined parameterization, since it is surjective and has a continuous inverse as we see below. However, computation by the infinite series is very slow, and hence, it is crucial to have an efficient algorithm for applications. In the next section, we will discuss the fast and explicit formula for the computation.

Although the above map ϕ is not one-to-one, we can compute its continuous inverse explicitly, thanks to the *Cartan decomposition theorem*. The inverse map ψ is given by

$$(2) \qquad \begin{array}{rrcl} \psi: & \mathrm{Aff}^+(3) & \to & \mathfrak{se}(3) \times \mathfrak{sym}(3) \\ & A & \mapsto & \log(A\,\iota(\sqrt{{}^t\hat{A}\hat{A}})^{-1}) \times \log(\sqrt{{}^t\hat{A}\hat{A}}). \end{array}$$

Note that ${}^t\hat{A}\hat{A}$ is symmetric positive definite so that the square root is uniquely determined and the logarithm is also well-defined (it is calculated by [5] and [4], for example). Note also that $A\,\iota(\sqrt{{}^t\hat{A}\hat{A}})^{-1}$ is an element in SE(3) and the logarithm is defined up to modulo 2π. We discuss the explicit formulae in the next section.

2. The parameterization algorithm

First, we consider how to compute (2). Note that ${}^t\hat{A}\hat{A}$ is a positive definite symmetric matrix so that it is diagonalized as

$$P \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} {}^tP$$

with some orthogonal matrix P and $\lambda_i > 0$. Then we can compute

$$\log(\sqrt{{}^t\hat{A}\hat{A}}) = P \begin{pmatrix} \log(\sqrt{\lambda_1}) & 0 & 0 \\ 0 & \log(\sqrt{\lambda_2}) & 0 \\ 0 & 0 & \log(\sqrt{\lambda_3}) \end{pmatrix} {}^tP.$$

Let $R = A\ \iota(\sqrt{{}^t\hat{A}\hat{A}})^{-1}$. Since $R \in \mathrm{SE}(3)$, we can write $R = \begin{pmatrix} \hat{R} & d_R \\ 0 & 1 \end{pmatrix}$. By mimicking the famous Rodrigues' formula [3] for the rotation matrices, we have

$$\log(R) = \begin{pmatrix} \hat{X} & l_X \\ 0 & 0 \end{pmatrix},$$

where $\hat{X} = \dfrac{\theta}{2\sin\theta}(R - {}^tR)$, $\theta = \cos^{-1}\left(\dfrac{\mathrm{Tr}(R) - 1}{2}\right)$, and

$$l_X = \left(I_3 - \frac{1}{2}\hat{X} + \frac{2\sin\theta - (1+\cos\theta)\theta}{2\theta^2 \sin\theta}\hat{X}^2 \right) d_R.$$

As we mentioned before, here we have indeterminacy of $\cos^{-1}$ up to modulo 2π. However, if we impose continuity, we can take one explicit choice. (An explicit code is given in [8].)

Next, we consider how to compute (1). For any symmetric matrix $Y \in \mathfrak{sym}(3)$, any matrix function can be computed using diagonalization. However, we introduce a faster algorithm to compute the exponential based on the *spectral decomposition* (see [11], for example). In applications, we have to compute (2) only once as pre-computation and (1) many times in real-time. Hence it is important to have a fast algorithm to compute the exponential maps.

Let $\lambda_1, \lambda_2, \lambda_3$ be the eigenvalues of $Y \in \mathfrak{sym}(3)$. They are the roots of the characteristic polynomial of Y:

$$\lambda^3 - \mathrm{Tr}(Y)\lambda^2 + \frac{\mathrm{Tr}(Y)^2 - ||Y||_F^2}{2}\lambda - \det(Y), \tag{3}$$

where $||Y||_F$ is the Frobenius norm of Y. Note that computing eigenvalues is much faster than computing diagonalization. Now we can compute the exponential of Y as a degree two polynomial of Y rather than the infinite Taylor series.

When all the three eigenvalues are same, put

$$a = \exp(\lambda_1), b = c = 0.$$

When two of them are same, say $\lambda_1 = \lambda_2$, put

$$s = \exp(\lambda_2)/(\lambda_2 - \lambda_3), t = \exp(\lambda_3)/(\lambda_2 - \lambda_3), a = s - t, b = t\lambda_2 - s\lambda_3, c = 0.$$

When all of them are distinct, put

$$\begin{aligned}
s &= \exp(\lambda_1)/(\lambda_1 - \lambda_2)(\lambda_1 - \lambda_3), \\
t &= \exp(\lambda_2)/(\lambda_2 - \lambda_3)(\lambda_1 - \lambda_2), \\
u &= \exp(\lambda_3)/(\lambda_2 - \lambda_3)(\lambda_3 - \lambda_1), \\
a &= s\lambda_2\lambda_3 - t\lambda_3\lambda_1 - u\lambda_1\lambda_2, \\
b &= -s(\lambda_2 + \lambda_3) + t(\lambda_3 + \lambda_1) + u(\lambda_1 + \lambda_2), \\
c &= s + t + u.
\end{aligned}$$

Then we have

$$\exp(Y) = aI_3 + bY + cY^2. \tag{4}$$

Finally, for $X = \begin{pmatrix} \hat{X} & l_X \\ 0 & 0 \end{pmatrix} \in \mathfrak{se}(3)$, again by mimicking Rodrigues' formula, we have

$$\exp(X) = \begin{pmatrix} \hat{R} & d \\ 0 & 1 \end{pmatrix},$$

where

$$\hat{R} = I_3 + \frac{\sin\theta}{\theta}\hat{X} + \frac{1-\cos\theta}{\theta^2}\hat{X}^2,$$

and

$$d = \left(I_3 + \frac{1-\cos\theta}{\theta^2}\hat{R} + \frac{\theta - \sin\theta}{\theta^3}\hat{R}^2 \right) l_X.$$

3. Deformer applications

Now we explain the algorithm of our deformers. The basic framework is the following: the input is:

- a target shape to be deformed,
- a set of affine transformations $\{A_i \in \mathrm{Aff}^+(3) \mid 1 \le i \le m\}$,
- and weight functions on the vertices (or the simplex) $\{w_i : V \to \mathbb{R} \mid 1 \le i \le m\}$, where V is the set of the vertices (or the simplex) of the target shape.

With the above data, we deform the given shape by the following recipe:

$$V \ni v \mapsto \sum_{i=1}^{m} \phi(w_i \psi(A_i))v, \tag{5}$$

where we think of the vertex positions $v \in \mathbb{R}^3$ as column vectors and the matrices multiply from the left. When we take V as the set of simplex, the above formula gives non-consistent map on the edges, and we need to patch them by certain energy minimizing technique such as ARAP (§3.3). Among the good properties of our parameterization is that $\sum_{i=1}^{m} \phi(w_i \psi(A_i))$ is rigid when A_i's are.

According to how users specify (2) and (3) above, we introduce the three deformers.

3.1. **Probe-based deformer.** Given a target shape and any number of "probes" which carry transform data. If probes are transformed by the user, the target shape will be deformed according to it. More precisely, each probe detects the affine map $A_i \in \mathrm{Aff}^+(3)$ which transforms it to the current position from the initial position. A vertex $v \in \mathbb{R}^3$ on the target shape is transformed by the equation (5), where the weights w_i's are either painted manually, or computed automatically from the distance between v and the probe location (see Figures 1 and 2).

3.2. **Cage-based deformer.** Given a target shape and a "cage" surrounding it. The cage can be any triangulated polyhedron wrapping the target shape. We want to deform the target shape by manipulating not directly on it but through proxy cage (see [7]).

Our parameterization can be used in this framework. We associate a tetrahedra to each face triangle by adding its normal vector. Then each face detects the affine map $A_i \in \mathrm{Aff}^+(3)$ which transforms the initial tetrahedra to the current tetrahedra. A vertex $v \in \mathbb{R}^3$ on the target shape is transformed by the equation (5), where the weights w_i's are either painted manually, or computed automatically from the distance between v and the center of the face. In automatic weight computation, it is better to set $w_i = 0$ when v sits in the outer half space of the i-th face (see Figure 3).

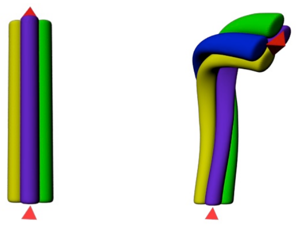


FIGURE 1. Left: set up initial positions of (red) probes. Right: the target shape is deformed according to user's manipulation of the probes.

FIGURE 2. Probe-based deformer can be used for the deformation of particles.

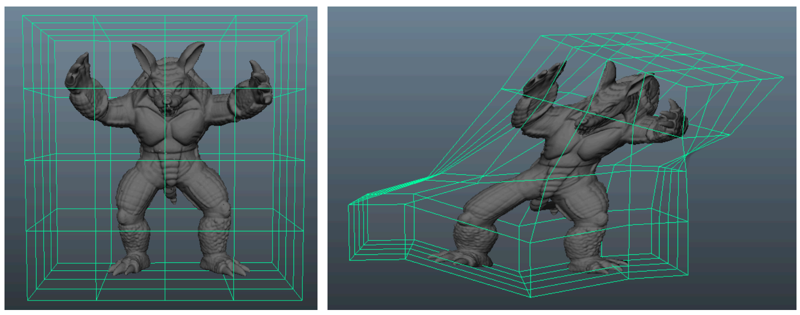


FIGURE 3. Left: initialize the target shape and the cage. Right: the target shape is deformed according to user's manipulation of the cage.

3.3. **Shape blender.** Given a target mesh V_0 and meshes to be blended $V_i \quad (1 \leq i \leq n)$. We assume that all the meshes are compatibly triangulated, i.e., a one-to-one correspondence for each pair of mesh is explicitly given.

A blended shape $V(w_1, \ldots, w_n)$ will be generated with respect to specified weights $\{w_i \in \mathbb{R} \mid 1 \leq i \leq n\}$ such that $V(w_1, \ldots, w_n) = V_0$ if $w_i = 0$ for any i, and $V(w_1, \ldots, w_n) = V_k$ if $w_i = \begin{cases} 1 & (i = k) \\ 0 & (i \neq k) \end{cases}$.

First, we associate for each face $f_{ij} \quad (1 \leq j \leq m)$ of V_i the unique affine transformation A_{ij} which maps the corresponding face f_{0j} with the unit normal vector to f_{ij} with the unit normal vector.

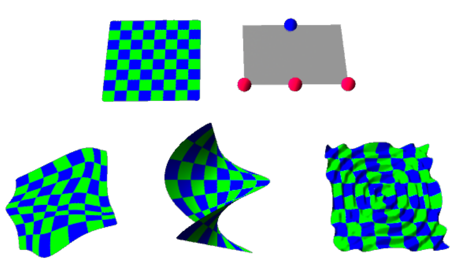
Then, we define

$$A'_j(w_1, \ldots, w_n) := \phi(\sum_{1 \leq i \leq n} w_i \psi(A_{ij})).$$

We cannot use those blended transformations as they are since they are not coherent on the edges. Finally, by minimizing a certain energy function (see [2], for example), we obtain a piecewise linear transformation $\{A_j(w_1, \ldots, w_n) \mid 1 \leq j \leq m\}$ and the blended shape is obtained as

$$V(w_1, \ldots, w_n) = \bigcup_{1 \leq j \leq m} A_j(w_1, \ldots, w_n) f_{0j}$$

(see Figures 4 and 5).


(a)

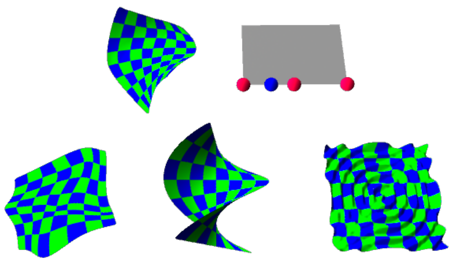
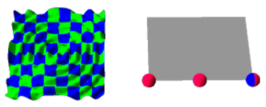

(b)

Figure 4. (a): upper-left is the target shape. The upper-right is a controller. The other three are the shapes to be blended. The blue ball corresponds to the target shape, and the left (resp., center, right) red ball corresponds to the left (resp., center, right) shape. (b): the target shape is deformed according to the weights specified through the ball controller.


(c)

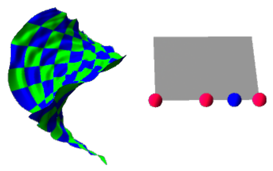

(d)

Figure 5. (c): the target shape matches the blended shape when the weights are $(0, 0, 1)$. (d): the weight can be outside the range of 0 to 1; this means extrapolation.

Acknowledgements

This work was supported by Core Research for Evolutional Science and Technology (CREST) Program "Mathematics for Computer Graphics" of Japan Science and Technology Agency (JST). The authors are grateful to A. Kimura, Y. Katsura and M. Salvati at OLM Digital, Inc., and B. Edlington at Toneplus Animation Studios, and Y. Mizoguchi, S. Yokoyama, H. Hamada, and K. Matsushita at Kyushu University for their valuable comments.

References

[1] M. Alexa, *Linear Combination of Transformations*, ACM Trans. Graph., 21(3), pp. 380–387, 2002.

[2] M. Alexa, D. Cohen-Or and D. Levin, *As-Rigid-As-Possible Shape Interpolation*, In Proceedings of the 27th annual conference on Computer graphics and interactive techniques, SIGGRAPH 2000, pp. 157–164, 2000.

[3] R. W. Brockett, *Robotic Manipulators and the Product of Exponentials Formula*, Mathematical Theory of Networks and Systems, Lecture Notes in Control and Information Sciences, 58, pp. 120–129, 1984.

[4] S. H. Cheng, N. J. Higham, C. S. Kenney and A. J. Laub, *Approximating the Logarithm of a Matrix to Specified Accuracy*, SIAM J. Matrix Anal. Appl., 22(4), pp. 1112–1125, 2001.

[5] E. D. Denman and A. N. Beavers, *The Matrix Sign Function and Computations in Systems*, Appl. Math. Comput., 2(1), pp. 63–94, 1976.

[6] J. E. Humphreys, *Introduction to Lie Algebras and Representation Theory*, Second printing, revised, Graduate Texts in Mathematics, 9, Springer-Verlag, New York, 1978.

[7] T. Ju, S. Schaefer and J. Warren, *Mean Value Coordinates for Closed Triangular Meshes*, ACM Trans. Graph., 24(3), pp. 561–566, 2005.

[8] S. Kaji, S. Hirose, H, Ochiai and K. Anjyo, *A Concise Parameterization of Affine Transformation*, in preparation.

[9] L. Kavan, S. Collins, J. Zara and C. O'Sullivan, *Geometric Skinning with Approximate Dual Quaternion Blending*, ACM Trans. Graph., 27(4), 105, 2008.

[10] J. P. Lewis, M. Cordner and N. Fong, *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, SIGGRAPH 2000, pp. 165–172, 2000.

[11] C. Moler and C. van Loan, *Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later*, SIAM Rev., 45(1), pp. 3–49, 2003.

[12] H. Ochiai, *Mathematical Formulation of Motion/Deformation and its Applications*, in this volume.