Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Smooth interpolation on homogeneous matrix groups for computer animation*

LI Jun[1,2], HAO Peng-wei[1,2]

([1]*Center for Information Science, Peking University, Beijing 100081, China*)

([2]*Department of Computer Science, Queen Mary, University of London, London, E1 4NS, UK*)

E-mail: {junjy; phao}@cis.pku.edu.cn; {junjy; phao}@dcs.qmul.ac.uk

Received Apr. 18, 2006; revision accepted Apr. 29, 2006

Abstract: Homogeneous matrices are widely used to represent geometric transformations in computer graphics, with interpolation between those matrices being of high interest for computer animation. Many approaches have been proposed to address this problem, including computing matrix curves from curves in Euclidean space by registration, representing one-parameter curves on manifold by rational representations, changing subdivisional methods generating curves in Euclidean space to corresponding methods working for matrix curve generation, and variational methods. In this paper, we propose a scheme to generate rational one-parameter matrix curves based on exponential map for interpolation, and demonstrate how to obtain higher smoothness from existing curves. We also give an iterative technique for rapid computing of these curves. We take the computation as solving an ordinary differential equation on manifold numerically by a generalized Euler method. Furthermore, we give this algorithm's bound of the error and prove that the bound is proportional to the shift length when the shift length is sufficiently small. Compared to direct computation of the matrix functions, our Euler solution is faster.

Key words: Computer animation, Spline and piecewise polynomial approximation
doi:10.1631/jzus.2006.A1168 **Document code:** A **CLC number:** TP39; O24

INTRODUCTION

Homogeneous matrices are matrices used to represent geometric transformations for homogeneous coordinates. Because they provide uniform representation for various transformations, they are widely used in the computer graphics community. As a homogeneous matrix corresponds to a geometric transformation, a curve on the set of homogeneous matrices can be taken as a piece of motion. Thus interpolation between those matrices is of high interest in many applications such as computer animation and robot motion design. Some subsets of the homogeneous matrices form special interesting groups with matrix multiplication, such as homogeneous $SO(3)$ and $SE(3)$. The set of all the non-degraded homogeneous matrices also forms a group.

There are now many literature on interpolation in those groups. In $SO(3)$, Shoemake (1985) introduced quaternions to represent rotations to the computer graphics community. Barr *et al*.(1992) proposed a method using quaternions to interpolate transforms with constraints of velocity. Kim *et al*.(1995) introduced a general framework to transform an interpolated curve in $\mathbb{R}^3$ into a quaternion curve. Park and Ravani (1997) presented an approximate analytical solution for smooth interpolation in $SO(3)$. Noakes (2004) gave a method to generate of Bézier curve on spherical manifolds and applied it to $SO(3)$.

There are also voluminous literature dealing with interpolation in the group of 3D rigid body transformation matrices, $SE(3)$. However, because of the lack of the bi-invariant Riemannian metrics on $SE(3)$ (Zefran *et al*., 1999), it is very difficult to find appropriate interpolated curves in $SE(3)$. The algebra of twists and Chasles' theory of screws are used in (Hunt,

* Project (No. 200038) partially supported by FANEDD, China

1978; McCarthy, 1990). However, none of those methods can generate a motion with physical meaning (Zefran *et al*., 1999). Zefran *et al*. discussed metrics and connections on *SE*(3) (Zefran *et al*., 1999), and proposed methods to find trajectories satisfying certain optimistic conditions in left-invariant Riemannian metrics (Zefran *et al*., 1998).

However, there is relatively less work about interpolation in general homogeneous matrix groups. Alexa (2002) tried to establish a framework to interpolate in Lie group by exponential map. Due to lack of dealing with the non-flat property of space, this method leaves much space for improvement when applied to some kind of rotational matrices (Bloom *et al*., 2004). Wallner and Dyn (2005) proposed an in-depth analysis on the convergence and C^1 property of subdivision schemes for curve generation on general manifolds. Hofer *et al*.(2004) proposed a method based on registration of feature points. Hofer and Pottmann (2004) and Pottmann and Hofer (2005) presented a variational approach for curve design on manifolds, gave the properties of those curves and proposed an algorithm to obtain them.

In this paper, we briefly introduce some basics of the Lie group and the interpolation on Lie group using exponential map. We also propose a method similar to (Kim *et al*., 1995), but dealing with matrices so that it can be more general. We point out that the scheme can be extended to obtain higher-order of continuity.

For fast online computation, we exploit the matrix commutativity and its exponent. The computation of the rational expression is taken as the Euler numerical solution of ordinary differential equation on manifold. We also give error analysis for this method.

The rest of this paper is organized as follows. In Section 2, we introduce some necessary geometric concepts on matrix groups, and the common way to interpolate between two elements in a matrix group. In Section 3, we present a quadratic method to obtain motion with continuous velocity. In Section 4, we extend our method to a cubic form. In Section 5, we develop fast numerical implementation of our analytical solution which is followed by analysis of its error bound. And finally we conclude our paper in Section 6.

EXPONENTIAL MAP AND LINEAR INTERPOLATION

A three dimensional non-degraded homogeneous matrix has the form

$$\boldsymbol{M}=\begin{bmatrix}\boldsymbol{Q} & \boldsymbol{d}\\ \boldsymbol{0} & 1\end{bmatrix},$$

where $\boldsymbol{Q}$ is a 3×3 invertible matrix, and $\boldsymbol{d}$ is a vector in $\mathbb{R}^3$. It is well-known that the set of all homogeneous matrices forms a matrix group. All matrix groups are Lie groups.

Problem modelling

Mathematically, our problem of interpolation between transformations is how to generate a curve passing through some given elements (matrices for key frames) in a matrix group.

Formally, the problem can be stated as: Given n+1 time points t_0, t_1, …, t_n, and n+1 key matrices in some matrix group $\boldsymbol{G}$, $\boldsymbol{A}_0$, $\boldsymbol{A}_1$, …, $\boldsymbol{A}_n$, find a continuous curve $N(t)$ in $\boldsymbol{G}$, so that

$$N(t_i)=\boldsymbol{A}_i,\ i=0,\ \ldots,\ n. \tag{1}$$

Naturally, we solve Eq.(1) piecewise by finding n segments $N(t_i)$, i=0, …, n−1, satisfying $N_i(0)=\boldsymbol{A}_i$, and $N_i(t_{i+1}-t_i)=\boldsymbol{A}_{i+1}$.

By simple variable substitution, we can convert every time interval into 1 without loss of generality. Then the problem is to find $N_i(t)$, so that

$$N_i(0)=\boldsymbol{A}_i, \tag{2}$$
$$N_i(1)=\boldsymbol{A}_{i+1}. \tag{3}$$

Interpolation using exponential map

1. Lie group and exponential map

Lie algebra is the tangent vector space of a Lie group at its neutral element. If $\boldsymbol{G}$ is a Lie group, and $\mathfrak{g}$ is its Lie algebra, the exponential map is a map: $\exp:\mathfrak{g}\to\boldsymbol{G}$. For a matrix group, we have (Godinho and Natário, 2004)

$$\exp(\boldsymbol{V})=\mathrm{e}^{\boldsymbol{V}}=\sum_{k=0}^{\infty}\frac{\boldsymbol{V}^k}{k!}. \tag{4}$$

With the formula above, exp maps a curve in Lie algebra into Lie group, which is the basic idea of using exponential map to interpolate between matrices.

2. Naive interpolation

As the tangent space is a vector space, and thus is

convenient for applying the many tools of curve generation in Euclidean space. For a Lie group $\boldsymbol{G}$, its elements can be associated with the elements in its Lie algebra easily. Thus it is natural to interpolate in $\mathfrak{g}$, and then map the curve back to $\boldsymbol{G}$.

For two elements A and B in matrix group $\boldsymbol{G}$, we first find the two corresponding elements $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$ in its Lie algebra $\mathfrak{g}$, so that $e^{v_A}=\boldsymbol{A}$ and $e^{v_B}=\boldsymbol{B}$. Then, in vector space $\mathfrak{g}$, we linearly link $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$ by $v(t)=(1-t)\boldsymbol{v}_A+t\boldsymbol{v}_B$. Finally we find the interpolation in $\boldsymbol{G}$ by $c(t)=e^{v(t)}$, i.e.

$$c(t)=e^{(1-t)\log \boldsymbol{A}+t\log \boldsymbol{B}}. \tag{5}$$

3. Problems with naive interpolation

Unfortunately, naive interpolation has the problems of distortion and global neighborhood error, caused by the map between spaces with different topologies (Bloom *et al*., 2004). We shall see this in the case of $SO(3)$.

4. Distortion

Because $\mathbb{R}^3$ is flat and $SO(3)$ is not flat, a straight line in $\mathbb{R}^3$ does not necessarily have an image of a geodesic in $SO(3)$ (Do Carmo, 1992). Fig.1 shows the distortion of the naive interpolation in $SO(3)$.

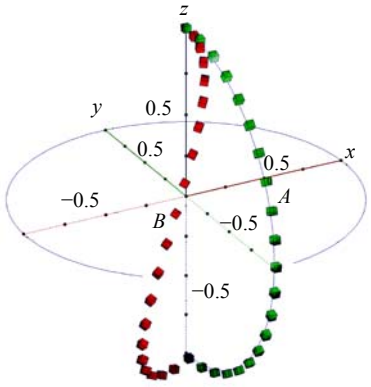

Fig.1 The curvature of $SO(3)$. This figure shows the distortion of naive interpolation from R_1 to R_2, with R_1 being a rotation near π about z-axis. $R_2=R_xR_1$, where R_x is a rotation near π about x-axis. A shows a correct trajectory, B is obtained by naive interpolation.

5. Global neighborhood error

Because the exponent map is not bijective, for one single rotation matrix in $SO(3)$, there are infinite vectors corresponding to it in $SO(3)$. On some occasions, there may not be 'natural' way to choose one of those vectors. Fig.2 demonstrates the global neighborhood errors. Because rotation $\boldsymbol{B}$ and $\boldsymbol{C}$ are identical orientation with different preimage for exponential map, the logarithmic algorithm may make an undesirable choice.

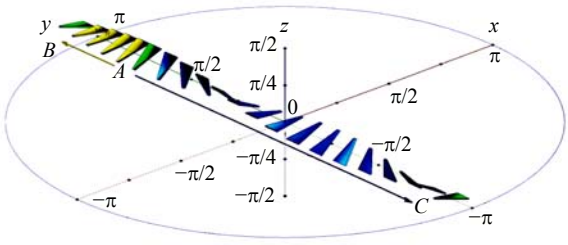

Fig.2 Global neighborhood error. A and B are two rotation matrices in $SO(3)$, C is another preimage of rotation matrix B in $SO(3)$. The trajectory from A to B is 'natural', but the naive interpolation method may give a trajectory from A to C.

Interpolation using exponential map with left translation

However, for a straight line that passes the origin in Lie algebra, the exponential curve is free from distortion in Lie group. Furthermore, it is a geodesic in $SO(3)$, say, the 'natural straight line' on manifold. For a Lie group, the image of a geodesic by left translation is also a geodesic.

In order to find the geodesic from $\boldsymbol{A}$ to $\boldsymbol{B}$ in $\boldsymbol{G}$, we first assume that it is the image of another geodesic under left translation $\boldsymbol{L}_A$. That 'another geodesic' is from $\boldsymbol{I}$ to $\boldsymbol{A}^{-1}\boldsymbol{B}$, whose image under left translation $\boldsymbol{L}_A$ is $\boldsymbol{B}$. Then we turn our problem to find a geodesic from $\boldsymbol{I}$ to $\boldsymbol{A}^{-1}\boldsymbol{B}$, where the exponential map works.

The new curve is

$$c(t)=\boldsymbol{A}e^{t\boldsymbol{\omega}_{\boldsymbol{A}^{-1}\boldsymbol{B}}}. \tag{6}$$

where $\boldsymbol{\omega}_{\boldsymbol{A}^{-1}\boldsymbol{B}}$ satisfies $e^{\boldsymbol{\omega}_{\boldsymbol{A}^{-1}\boldsymbol{B}}}=\boldsymbol{A}^{-1}\boldsymbol{B}$.

For $SO(3)$, this scheme also deals with the problem of using the logarithm image by setting a reasonable limitation to the rotation in that the rotation angles should be less than π between every two adjacent key matrices.

Linear interpolation

We present below a solution to the linear interpolation problem in Eq.(2) and Eq.(3).

The curve $N(t)$ consists of n segments of curves that link pairs $(\boldsymbol{A}_0,\boldsymbol{A}_1)$, $(\boldsymbol{A}_1,\boldsymbol{A}_2)$, …, $(\boldsymbol{A}_{n-1},\boldsymbol{A}_n)$. As mentioned above, we assume that the length of each time interval is 1. Then the curve starts from t=0, and t_i=i.

Formally,

$$N(t) = \sum_{i=0}^{n-1} \delta_i(t) N_i(t-i), \tag{7}$$

where the configuring weights are

$$\delta(t) = \begin{cases} 1, & t \in [i, i+1), \\ 0, & \text{otherwise.} \end{cases}$$

The (i+1)th segment $N_i(t)$ is interpolated between $(\boldsymbol{A}_i,\boldsymbol{A}_{i+1})$ with t from 0 to 1. We obtain $N_i(t)$ by left translating $M_i(t)$, where $M_i(t)$ is interpolated between $(\boldsymbol{I}, \boldsymbol{A}_i^{-1}\boldsymbol{A}_{i+1})$.

$$N_i(t) = \boldsymbol{A}_i M_i(t),\ i = 0,\ ...,\ n-1, \tag{8}$$

where

$$M_i(t) = \mathrm{e}^{t\,\boldsymbol{B}_i},\ t \in [0,1],\ i = 0,\ \ldots,\ n-1, \tag{9}$$

and

$$\boldsymbol{B}_i = \log(\boldsymbol{A}^{-1}\boldsymbol{A}_{i+1}),\ i = 0,\ ...,\ n-1. \tag{10}$$

QUADRATIC INTERPOLATION

By extending the above methods, we can obtain a curve in homogeneous matrix group passing a sequence of given matrices. This curve represents a continuous motion. However, for a more pleasing visual effect, we also wish that the velocity is also continuous.

Park and Ravani (1997) presented a technique on $SO(3)$, which added a quadratic and a cubic term to the linear interpolation equation in a natural way. As shown below, this can yield an approximation. While Kim *et al*.(1995) proposed a framework to map a curve in $\mathbb{R}^3$ to a quaternion curve. Wallner and Dyn (2005) discussed smooth properties of subdivision schemes on manifolds which were derived from similar schemes on Euclidean space. However, they mainly worked in the extrinsic space in which the matrix group is embedded, thus less light was cast on properties of the generated curves in their intrinsic manifold.

Here, we study the velocity of a curve in a matrix group. With this knowledge, we develop a technique to extend the linear interpolation to a quadratic one so that the interpolation curve is C^1. In the next section, we will show the extension can be reused to achieve higher order of continuity.

The C^1 conditions

We define velocity, and give the equation in terms of our interpolation problem.

1. Velocity

For a curve $c(t)$ passing p in a matrix group $\boldsymbol{G}$, let us agree that $c(0)$=p. Its velocity at $c(0)$ is determined by a left invariant vector field X, where X satisfies $X(p) = \dot{c}(0)$. More specifically, $v_c(0)$=X_e, X_e is the value of X at e. Formally (Do Carmo, 1992),

$$v_c(t) = [c(t)]^{-1}\dot{c}(t). \tag{11}$$

2. Equations of the conditions

Besides Eq.(2) and Eq.(3), we add a new condition for continuity of velocity

$$\dot{N}_i(t) = \dot{N}_{i+1}(0). \tag{12}$$

By substituting $N(t)$ for $c(t)$ in Eq.(11), we can see Eq.(12) equals to $V_i(1)$=$V_{i+1}(0)$, where $V_i(t)$ is the velocity of $N_i(t)$.

Quadratic forms

Using Eq.(4), we have

$$\frac{\mathrm{d}\mathrm{e}^{f(t)\boldsymbol{G}}}{\mathrm{d}t} = f'(x)\mathrm{e}^{f(x)\boldsymbol{G}}\boldsymbol{G}. \tag{13}$$

Comparing Eq.(9), Eq.(8) and Eq.(13), and by Eq.(11), each segment $N_i(t)$ in Eq.(8) has a constant velocity $\boldsymbol{B}_i$. As $\boldsymbol{B}_i$ is determined by Eq.(10), it is natural to add a quadratic item to $M_i(t)$ in Eq.(9) to obtain a C^1 curve.

1. Quadratic term

Notice that in Eq.(4), a matrix and its exponent are commutative, which means that we have an alternative form of Eq.(13)

$$\frac{\mathrm{d}\mathrm{e}^{f(t)\boldsymbol{G}}}{\mathrm{d}t} = f'(x)\boldsymbol{G}\mathrm{e}^{f(x)\boldsymbol{G}}.$$

If we let our quadratic form

$$M_i(t) = \mathrm{e}^{t\boldsymbol{B}_i}\mathrm{e}^{t^2\boldsymbol{C}_i},$$
$$\text{where } t\in[0,1],\ i=0,\ldots,n-1. \tag{14}$$

We have a closed form of its derivative, so that we can figure out the parameter $\boldsymbol{B}_i$ and $\boldsymbol{C}_i$ analytically according to the continuity constraint.

2. Quadratic solution

The derivative for $M_i(t)$ in Eq.(14) is

$$\begin{aligned}\dot{M}_i(t) &= \frac{\mathrm{d}\mathrm{e}^{t\boldsymbol{B}_i}}{\mathrm{d}t}\mathrm{e}^{t^2\boldsymbol{C}_i} + \mathrm{e}^{t\boldsymbol{B}_i}\frac{\mathrm{d}\mathrm{e}^{t^2\boldsymbol{C}_i}}{\mathrm{d}t}\\ &= \mathrm{e}^{t\boldsymbol{B}_i}\boldsymbol{B}_i\mathrm{e}^{t^2\boldsymbol{C}_i} + 2t\mathrm{e}^{t\boldsymbol{B}_i}\mathrm{e}^{t^2\boldsymbol{C}_i}\boldsymbol{C}_i\\ &= \boldsymbol{B}_i M_i(t) + 2tM_i(t)\boldsymbol{C}_i,\end{aligned} \tag{15}$$

Condition Eq.(2) is met automatically at t=0 regardless of our choice of $\boldsymbol{B}_i$ and $\boldsymbol{C}_i$.

For condition Eq.(3), we can choose $\boldsymbol{C}_i$ so that

$$\mathrm{e}^{\boldsymbol{B}_i}\mathrm{e}^{\boldsymbol{C}_i} = \boldsymbol{A}_i^{-1}\boldsymbol{A}_{i+1},$$

then

$$\boldsymbol{C}_i = \log((\mathrm{e}^{\boldsymbol{B}_i})^{-1}\boldsymbol{A}_i^{-1}\boldsymbol{A}_{i+1}). \tag{16}$$

Finally, from Eq.(15), we meet condition Eq.(12) by choosing $\boldsymbol{B}_{i+1}$ for the (i+1)th segment so that

$$\boldsymbol{B}_{i+1} = \boldsymbol{A}_{i+1}^{-1}\boldsymbol{A}_i\boldsymbol{B}_i\boldsymbol{A}_i^{-1}\boldsymbol{A}_{i+1} + 2\boldsymbol{C}_i. \tag{17}$$

CUBIC INTERPOLATION

In the above discussion, we do quadratic Hermit interpolation for each segment of the curve. However we can only specify the initial velocity for the first segment: If we specify $\dot{M}_0(0)$, then $\boldsymbol{B}_0$ is determined, and then by condition Eq.(16), $\boldsymbol{C}_0$ is determined, and furthermore, by condition Eq.(17), $\boldsymbol{B}_1$ is determined, then $\boldsymbol{C}_1$, then $\boldsymbol{B}_2$, …, till the last segment. It is undesirable because it makes not only the local adjustment of our curve difficult, but also the velocity at other points uncontrollable. We can see this in Figs.3a~3c.

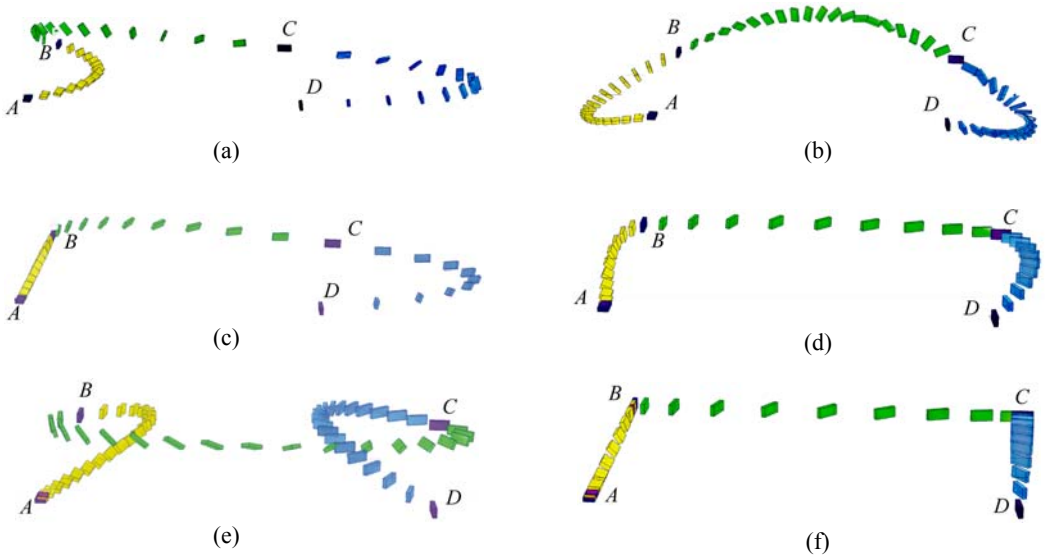

Figure	Interpolation form	Velocity $\boldsymbol{A}$	Velocity $\boldsymbol{B}$	Velocity $\boldsymbol{C}$	Velocity $\boldsymbol{D}$
(a)	Quadratic	$0.9\log(\boldsymbol{A}^{-1}\boldsymbol{C})$	N/A	N/A	N/A
(b)	Quadratic	$-0.9\log(\boldsymbol{A}^{-1}\boldsymbol{C})$	N/A	N/A	N/A
(c)	Quadratic	$\log(\boldsymbol{A}^{-1}\boldsymbol{B})$	N/A	N/A	N/A
(d)	Cubic	$\log(\boldsymbol{A}^{-1}\boldsymbol{B})/3$	$\log(\boldsymbol{A}^{-1}\boldsymbol{C})/3$	$\log(\boldsymbol{B}^{-1}\boldsymbol{D})/3$	$\log(\boldsymbol{C}^{-1}\boldsymbol{D})/3$
(e)	Cubic	$\log(\boldsymbol{A}^{-1}\boldsymbol{B})/3$	$-\log(\boldsymbol{A}^{-1}\boldsymbol{C})/3$	$-\log(\boldsymbol{B}^{-1}\boldsymbol{D})/3$	$\log(\boldsymbol{C}^{-1}\boldsymbol{D})/3$
(f)	Cubic	0	0	0	

Fig.3 *A*, *B*, *C* and *D* indicate four configurations of the model at t=0, 1, 2 and 3. In the table below the figures, *A*, *B*, *C* and *D* denote the corresponding transformation matrices for these configurations.

The C^2 conditions

Now we wish to be able to specify the velocity of our curve at every time point, thus we change the condition in Eq.(12) to

$$\dot{N}_i(0) = \boldsymbol{A}_i \boldsymbol{V}_i, \tag{18}$$

$$\dot{N}_i(1) = \boldsymbol{A}_{i+1} \boldsymbol{V}_{i+1}, \tag{19}$$

where i=0,…,n−1, $\boldsymbol{V}_{i+1}$ can be specified freely. For time points i=0,…,n−2, we have

$$\dot{N}_i(1) = \dot{N}_{i+1}(0) = \boldsymbol{A}_{i+1} \boldsymbol{V}_{i+1}, \tag{20}$$

which means that the velocity is always continuous and we can freely specify the velocity at every time point.

Cubic forms

To do this, we need another free term to obtain that flexibility, the cubic term. Our new form is

$$M_i(t) = \mathrm{e}^{t\boldsymbol{B}_i} \mathrm{e}^{t^2 \boldsymbol{C}_i} \mathrm{e}^{(t^3 - t^2)\boldsymbol{D}_i}. \tag{21}$$

1. Remark

We use $f(t)=t^3-t^2$ as the weight function for the cubic term because

(1) $f(0)=f(1)=0$, then we obtain closed form of the derivative of Eq.(21) at the beginning and the end of every segment of the curve.

(2) $f'(0)=0$, then we can choose a B_i to specify the initial speed for a segment of the curve as before.

(3) $f'(1)=1$, then we can control the velocity of the segment at its end point by specifying D_i.

Therefore we now have

$$M_i(0) = \boldsymbol{I}, \tag{22}$$

$$M_i(1) = \mathrm{e}^{\boldsymbol{B}_i} \mathrm{e}^{\boldsymbol{C}_i}, \tag{23}$$

$$\dot{M}_i(0) = \boldsymbol{B}_i, \tag{24}$$

$$\dot{M}_i(1) = \boldsymbol{B}_i \boldsymbol{A}_i^{-1} \boldsymbol{A}_{i+1} + 2\mathrm{e}^{\boldsymbol{B}_i} \mathrm{e}^{\boldsymbol{C}_i} \boldsymbol{C}_i + \boldsymbol{A}_i^{-1} \boldsymbol{A}_{i+1} \boldsymbol{D}_i. \tag{25}$$

2. Cubic solution

We can specify the initial velocity of each segment of the curve by specifying $\boldsymbol{B}_i$

$$\boldsymbol{B}_i = \dot{M}_i(0) = \boldsymbol{A}_i^{-1} \boldsymbol{A}_i \dot{M}_i(0) = (N(t_i))^{-1} N(t_i) = v_N(t_i). \tag{26}$$

For $\boldsymbol{C}_i$, we have

$$\boldsymbol{C}_i = \log(\mathrm{e}^{\boldsymbol{B}_i} \boldsymbol{A}_i^{-1} \boldsymbol{A}_{i+1}), \tag{27}$$

so that when substituting Eq.(23) in Eq.(8), Eq.(3) can be satisfied.

For $\boldsymbol{D}_i$, we have

$$\boldsymbol{D}_i = \boldsymbol{B}_{i+1} - \boldsymbol{A}_{i+1}^{-1} \boldsymbol{A}_i \boldsymbol{B}_i \boldsymbol{A}_i^{-1} \boldsymbol{A}_{i+1} - 2\boldsymbol{C}_i, \tag{28}$$

so that Eq.(20) can be satisfied.

Some examples with different configurations are given in Figs.3d~3f.

Extension of existing interpolation equations

In fact, by following the procedure of adding free terms to existing interpolation equations, we can obtain higher-order smoothness. For example, if we have an interpolation equation $N_i(t)=\mathrm{e}^{f_0^i(t)P_0^i}\mathrm{e}^{f_1^i(t)P_1^i}\ldots\mathrm{e}^{f_{m-1}^i(t)P_{m-1}^i}$, and $N(t)$ has $(n-1)$th order of continuity, we can obtain nth order continuity by adding another item $f_m^i(t)P_m^i$. As long as we set $f_m^i(0)=f_m^i(1)=f_m^{i\prime}(0)=f_m^{i\prime}(1)=\ldots=f_m^{i(n)}(0)=0$, $f_m^{i(n)}(1)=1$, we only need to figure out a value for the new parameter P_m^i to make the higher derivative $N_i^{(n)}(t)$ at t=1 meet the requirement that it equals to $N_{i+1}^{(n)}(0)$, the same order derivative at the start point of the next segment. The latter is determined by the existing parameters $f_j^i(t)$ and P_j^i. As to the lower order derivatives of the new form, from the requirement of $f_m^i(t)$, it is obvious that the forms of the lower derivatives of the new form are the same as those in the original form. Thus we can keep the condition for lower order derivatives satisfied without changing existing parameters.

GENERALIZED EULER COMPUTATION

In practical usage of matrix interpolation, say, generating an animation clip between two given key frames, it is often needed to compute hundreds or even more than one thousand intermediate frames in one segment of our curve. And the computation is often taken from the beginning of the segment to the

end by proceeding at a tiny time shift Δt in each step. Thus if we can figure out the matrix for the 'next' frame $M(t_0+\Delta t)$ from the one for 'this' frame $M(t_0)$ in a faster way than direct computing $M(t_0)$ using matrix exponent, it will make our interpolation more applicable. We shall show it is possible with the help of $\dot{M}(t_0)$.

In fact, there exists an intuitive way of achieving this: $M(t_0+\Delta t)\approx M(t_0)+\Delta\dot{M}(t_0)$. The problem is actually to find $\dot{M}(t_0)$. We illustrate this computation below by applying it to our cubic interpolation.

In $\mathbb{R}$, if we wish to find the value of a C^1 function $f(t)$ at $t_0+\Delta t$, and we know some initial value at t_0, $f(t_0)$, the derivative of f for any t, $f'(t)$, then we have

$$\begin{aligned} f(t_0+\Delta t) &= f(t_0)+\int_{t_0}^{t_0+\Delta t}\dot{f}(u)\mathrm{d}u \\ &= f(t_0)+\Delta t\dot{f}(t_0+\alpha\Delta t),\ \alpha\in[0,1]. \end{aligned} \tag{29}$$

If Δt is small enough, we can estimate the value of f at $t+\Delta t$

$$f(t_0+\Delta t)\approx f(t_0)+\Delta t\dot{f}(t_0). \tag{30}$$

In our interpolation problem, with known B, C and D and initial value of $M(0)=I$, we need to find a curve segment $M(t):[0,1]\to\boldsymbol{G}$. (Note that we omit the index here.)

Analogous to Eq.(30) in elementary calculus, we have

$$M(t_0+\Delta t)=M(t_0)+\Delta t\dot{M}(t_0+\alpha\Delta t),\ \alpha\in[0,1]. \tag{31}$$

When Δt is small enough, we can obtain $M(t_0+\Delta t)$ approximately by

$$M(t_0+\Delta t)\approx M(t_0)+\Delta t\dot{M}(t_0). \tag{32}$$

Fast computation of Eq.(32) depends on the assumption that we can compute $\dot{M}(t)$ easily for any t. We are now finding the derivative for Eq.(21). From Eq.(15) we find that for any curve of the form

$$c(t)=\mathrm{e}^{f(t)\boldsymbol{B}}\mathrm{e}^{g(t)\boldsymbol{C}}, \tag{33}$$

we have its derivative

$$\dot{c}(t)=f'(t)\boldsymbol{B}c(t)+g'(t)c(t)\boldsymbol{C}. \tag{34}$$

Then we rewrite $M(t)$ in Eq.(21) as

$$M(t)=M_{\boldsymbol{BC}}(t)M_{\boldsymbol{D}}(t), \tag{35}$$

where

$$M_{\boldsymbol{BC}}(t)=\mathrm{e}^{t\boldsymbol{B}}\mathrm{e}^{t^2\boldsymbol{C}},\ M_{\boldsymbol{D}}(t)=\mathrm{e}^{t^2(t-1)\boldsymbol{D}}.$$

If we have $M_{\boldsymbol{BC}}(t)$ at some time t_0, then we can approximate its value with a small time shift Δt

$$M_{\boldsymbol{BC}}(t_0+\Delta t)\approx M_{\boldsymbol{BC}}(t_0)+\Delta t\dot{M}_{\boldsymbol{BC}}(t_0), \tag{36}$$

where by Eq.(34)

$$\dot{M}_{\boldsymbol{BC}}(t_0)=\boldsymbol{B}M_{\boldsymbol{BC}}(t_0)+2t_0M_{\boldsymbol{BC}}(t_0)\boldsymbol{C}. \tag{37}$$

By Eq.(36) and Eq.(37), we obtain

$$\begin{aligned} M_{\boldsymbol{BC}}(t_0+\Delta t)\approx M_{\boldsymbol{BC}}(t_0)+\Delta t[&\boldsymbol{B}M_{\boldsymbol{BC}}(t_0) \\ &+2t_0M_{\boldsymbol{BC}}(t_0)\boldsymbol{C}]. \end{aligned} \tag{38}$$

Similarly,

$$M_{\boldsymbol{D}}(t_0+\Delta t)\approx M_{\boldsymbol{D}}(t_0)+\Delta t\dot{M}_{\boldsymbol{D}}(t_0). \tag{39}$$

By Eq.(13), we can find $\dot{M}_{\boldsymbol{D}}(t)$ easily

$$\dot{M}_{\boldsymbol{D}}(t_0)=(3t_0^2-2t_0)M_{\boldsymbol{D}}(t_0)\boldsymbol{D}. \tag{40}$$

Thus we have

$$M_{\boldsymbol{D}}(t_0+\Delta t)\approx M_{\boldsymbol{D}}(t_0)+\Delta t((3t_0^2-2t_0)M_{\boldsymbol{D}}(t_0)\boldsymbol{D}). \tag{41}$$

In this way, we can 'move forward' a small step along $M_i(t)$

$$M_i(t_0+\Delta t)=M_{\boldsymbol{BC}}(t_0+\Delta t)M_{\boldsymbol{D}}(t_0+\Delta t), \tag{42}$$

by using only 4 matrix multiplications.

Computational error

For analyzing the error of the computation above, we first study that of the Euler solution of a real ODE. For any C^1 function $f(x)$ defined on $[a,b]$, given $f'(x)=g(f(x))$ we can calculate its value at $x\in[a,b]$ as follows:

Algorithm 1

1. Let $f_c(t)\leftarrow f(a)$
2. Choose a $\Delta t=(x-a)/n$, n is an integer
3. Compute $f_c'(t)=g(f_c(t))$
4. Compute $f_c(t+\Delta t)=f_c(t)+\Delta t f_c'(t)$
5. Let $t\leftarrow t+\Delta t$
6. Goto 3, unless $t=x$

The sufficient condition of the convergence of this algorithm is that the function $f(x)$ satisfies Lipschitz condition that there exists an L so that $|g(y_1)-g(y_2)|\leq L|y_1-y_2|$. However, to our question we can assume that g has the second derivative in (a,b).

Proposition 1 If we calculate the value of a function $f(x)$ at $x\in[a,b]$ by Algorithm 1, we shall get a result $f_c(x)=f(x)+e(x)$, where $e(x)$ denotes the error. $|e(x)|$ has an upper bound that is proportional to the step size Δt in Algorithm 1, if following conditions are satisfied:

(1) $f(x)$ is C^1 in $[a,b]$ and has second derivative in (a,b);

(2) $g(x)$ is C^1 in $[a,b]$ and has second derivative in (a,b);

(3) The step size is sufficiently small.

Proof According to Algorithm 1, at the kth step, we calculate $f_c(a+(k+1)\Delta t)$ from $f_c(a+k\Delta t)$. We denote the error before kth step e_k, so we begin with $f_c(a)=f(a)$ and $e_0=0$.

$$\begin{aligned}
&f_c(a+(k+1)\Delta t)\\
&=f_c(a+k\Delta t)+\Delta t g(f_c(a+m\Delta t))\\
&=f(a+k\Delta t)+e_k+\Delta t g(f(a+k\Delta t)+e_k)\\
&=f(a+k\Delta t)+e_k+\Delta t[g(f(a+k\Delta t))\\
&\quad+e_k g'(f(a+k\Delta t))+\frac{e_k^2}{2}g''(f(a+k\Delta)+\beta_k)] \qquad (43)\\
&=f(a+k\Delta t)+\Delta t f'(a+k\Delta t)+e_k\\
&\quad+\Delta t e_k g'(a+k\Delta t)+\Delta t\frac{e_k^2}{2}g''(f(a+k\Delta t)+\beta_k)\\
&=f(a+(k+1)\Delta t)-\frac{\Delta t^2}{2}f''(a+k\Delta t+\alpha_k)+e_k\\
&\quad+\Delta t e_k g'(a+k\Delta t)+\Delta t\frac{e_k^2}{2}g''(f(a+k\Delta t)+\beta_k)
\end{aligned}$$

From Eq.(43), we can obtain the upper bound of the increase of the error at the kth step.

$$\begin{aligned}
e_{k+1}&=e_k-\frac{\Delta t^2}{2}f''(a+k\Delta t+\alpha_k)+\Delta t e_k g'(a+k\Delta t)\\
&\quad+\Delta t\frac{e_k^2}{2}g''(f(a+k\Delta)+\beta_k), \qquad (44)\\
|e_{k+1}|&\leq|e_k|+D\Delta t^2+\Delta t E|e_k|+\Delta t F|e_k^2|,
\end{aligned}$$

where D is the upper bound of $|f''|$, E is that of $|f'|$, and F, $|g''|$.

If we have

$$|e_k|\leq 1 \text{ (or any other constant)}, \qquad (45)$$

and let

$$p=1+\Delta t C, \qquad (46)$$

$$q=\Delta t^2 D, \qquad (47)$$

$$C=E+F, \qquad (48)$$

then from Eq.(44), we have

$$|e_{k+1}|\leq p|e_k|+q, \qquad (49)$$

and then extend it recursively

$$\begin{aligned}
|e_{k+1}|&\leq p|e_k|+q\leq p(p|e_{k-1}|+q)+q\\
&\quad\cdots\\
&\leq p(p(\dots p(pe_0+q)+q\dots)+q)+q \qquad (50)\\
&=p^{k+1}e_0+(p^k+p^{k-1}+\dots+p+1)q\\
&=\frac{1-p^{k+1}}{1-p}q=\frac{(1+\Delta tC)^{k+1}-1}{C}\Delta tD.
\end{aligned}$$

From Eq.(50), for a given Δt, the upper bound of the error increases monotonously with k. So if a Δt makes $|e_{k+1}|\leq 1$, for any m, $m=0,\dots,k$, we have $e_m\leq 1$, thus the assumption Eq.(45) can be satisfied.

And thus given the step size, the global bound of the error at any $t\in[a,b]$ is the bound at the end point $t=b$. We write

$$|e_N|\leq\frac{1-p^N}{1-p}q=\frac{(1+\Delta tC)^{\frac{b-a}{\Delta t}}-1}{C}\Delta tD. \qquad (51)$$

When $b-a$ is a constant, and Δt is small enough so that $(1+\Delta tC)^{\frac{b-a}{\Delta t}}\to e^{C(b-a)}$, it is approximately proportional to Δt; when Δt is a sufficiently small constant, it is approximately proportional to e^{b-a}.

For sufficiently small ε, given $\Delta t<\varepsilon$ the bound Eq.(51) would be less than $\bar{C}\Delta t$, where $\bar{C}$ is a constant determined only by C, D, a and b. It completes the proof. □

Remark If we have $g''\equiv 0$ then we do not need the assumption Eq.(45).

For a matrix function

$$M(t)=\begin{bmatrix} M_{11}(t) & M_{12}(t) & \cdots & M_{1n}(t) \\ M_{21}(t) & M_{22}(t) & \cdots & M_{2n}(t) \\ \vdots & \vdots & \ddots & \vdots \\ M_{n1}(t) & M_{n2}(t) & \cdots & M_{nn}(t) \end{bmatrix},$$

whose derivative satisfies $M'(t)=G(M(t))$, we can take an element $M_{ij}(t)$ of the matrix function to see what will happen when Algorithm 1 is applied. $M_{ij}(t)$ is a real function.

$$\begin{aligned} &\bar{M}_{ij}(a+(k+1)\Delta t) \\ &=\bar{M}_{ij}(a+k\Delta t)+\Delta t G_{ij}(M(a+k\Delta t)) \\ &=M_{ij}(a+k\Delta t)+E_{ij}^{k}+\Delta t G_{ij}(M(a+k\Delta t)+\boldsymbol{E}^{k}), \end{aligned} \tag{52}$$

where $\bar{M}(t)$ represents the numerical result of the function $M(t)$, $\boldsymbol{E}^k$ is the error matrix before the kth step. From Eq.(43), the bound of E_{ij}^k should have similar form to that in Eq.(51), provided conditions in Proposition 1 hold for $M_{ij}(t)$.

However, in the matrix version of Algorithm 1, $\boldsymbol{G}_{ij}$ has a matrix as its input variable. So we must modify the condition about $g(x)$ to: $\partial^2\boldsymbol{G}_{ij}/(\partial M_{pq}\partial M_{rs})$ exists in (a,b) for any i, j, p, q, r, s in $1\ldots n$. Note that this condition implies that $\partial\boldsymbol{G}_{ij}/\partial M_{pq}$ is continuous in $[a,b]$ for any i, j, p, q in $1\ldots n$.

Having this condition satisfied, if we analyze the relationship between the bound of E_{ij}^{k+1} and E_{ij}^k as in Eq.(50), we will have similar result as in Proposition 1 for computing $M_{ij}(t)$ by Algorithm 1. The whole matrix will definitely converge in the sense of any kind of norm if all of its elements converge.

It is easy to see that the condition for $\boldsymbol{G}$ holds for Eq.(37). So our method is valid.

Fig.4 shows us the error at each t in each segment in our experiment. Fig.5 illustrates the greatest value of the error in the third segment according to varying Δt.

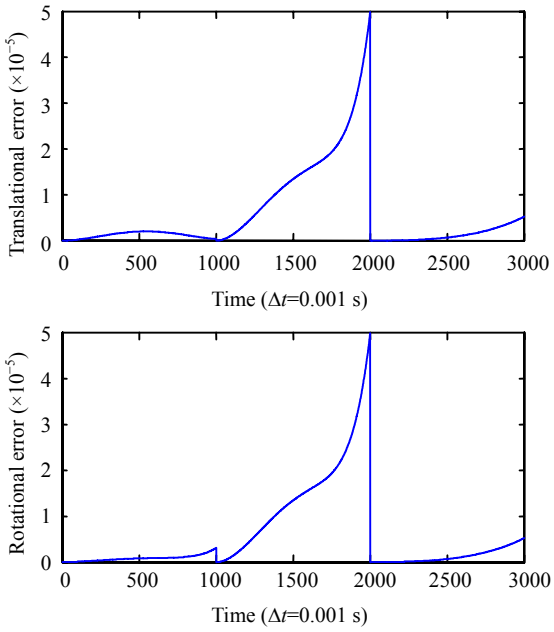

Fig.4 Approximation error of our method. The three-segment motion is the one we demonstrated in Fig.3. This figure illustrates the error of our numerical cubic method with different Δt in Eq.(42). For the measurement of error, we use the Riemannian metric for *SE*(3) (Zefran *et al*., 1999).

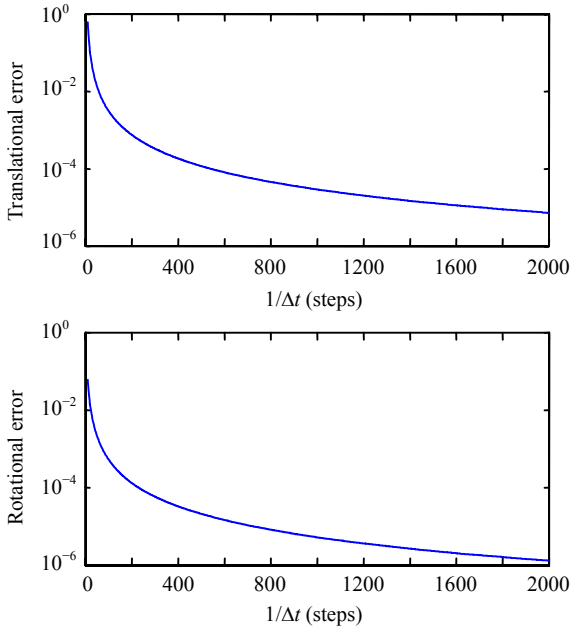

Fig.5 Experimental up-limit for the error. The three-segment motion is the one we demonstrated in Fig.3. This figure illustrates the error at the last moment in the last segment, which we suppose to be the greatest value in that segment (The value at 3000 in Fig.4). And that value decreases approximately with $1/\Delta t$. The metric is the same as that in Fig.4.

CONCLUSION

By using exponential map, we present the quadratic and cubic forms for animation interpolation between general homogeneous matrices. In our discussion, the development of quadratic and cubic interpolation in a matrix group also provides a basis for finding other forms of C^1 interpolation in a matrix group.

By taking the generation of the interpolated curve as solving a differential equation on manifolds, we propose a generalized Euler solution, and take advantage of the fact that there exists a closed form of $\dot{M}(t)$ with respect to known components of $M(t)$. Thus we achieve a lower computational complexity by avoiding the computation of matrix exponent at every step of the motion generation. We also have analyzed the convergence of the Euler solution. We have not precisely analyzed its computational complexity, but in our experiments, it is usually about twice faster than the implementation with the direct analytical form of matrix exponent.

In the future, it would be a possible research direction to make the methods with other physical equivalence such as acceleration by taking Riemannian connection on the matrix group into consideration.

Another sophisticated example for 19 given key matrices is illustrated in Fig.6.

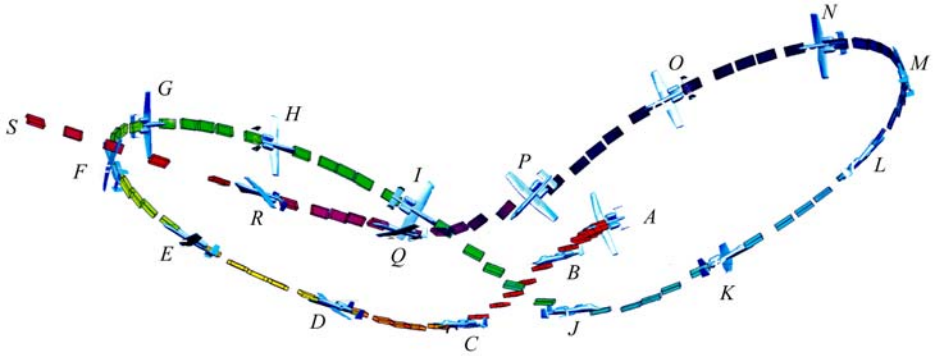

Fig.6 The interpolation result for a sequence of key matrices given at *A* to *S*, implemented with our cubic method

References

Alexa, M., 2002. Linear Combination of Transformations. Proceedings of SIGGRAPH'02, p.380-387.

Barr, A.H., Currin, B., Gabriel, S., Hughes, J.F., 1992. Smooth interpolation of orientations with angular velocity constraints using quaternions. *Computer Graphics (SIGGRAPH'92)*, **26**(2):313-320. [doi:10.1145/142920.134086]

Bloom, C., Blow, J., Muratori, C., 2004. Errors and Omissions in Marc Alexa's "Linear Combination of Transformations". http://www.cbloom.com/3d/techdocs/lcot_errors.pdf.

Do Carmo, M.P., 1992. Riemannian Geometry. Birkhauser, Springer, Boston, MA.

Godinho, L., Natário, J., 2004. An Introduction to Riemannian Geometry with Applications. Http://www.math.ist.utl.pt/~lgodin/.

Hofer, M., Pottmann, H., 2004. Energyminimizing splines in manifolds. *Transactions on Graphics*, **23**(3):284-293. [doi:10.1145/1015706.1015716]

Hofer, M., Pottmann, H., Ravani, B., 2004. From curve design algorithms to the design of rigid body motions. *The Visual Computer*, **20**(5):279-297. [doi:10.1007/s00371-003-0221-3]

Hunt, K.H., 1978. Kinematic Geometry of Mechanisms. Oxford University Press.

Kim, M.J., Myung-Soo, K., Shin, S.Y., 1995. A General Construction Scheme for Unit Quaternion Curves with Simple High Order Derivatives. Proc. of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH'95, p.369-376. [doi:10.1145/218380.218486]

McCarthy, J.M., 1990. Introduction to Theoretical Kinematics. MIT Press, Cambridge MA.

Noakes, L., 2004. Spherical Splines.

Park, F.C., Ravani, B., 1997. Smooth invariant interpolation of rotations. *ACM Transactions on Graphics (TOG)*, **16**(3):277-295. [doi:10.1145/256157.256160]

Pottmann, H., Hofer, M., 2005. A variational approach to spline curves on surfaces. *Computer Aided Geometric Design*, **22**(7):693-709. [doi:10.1016/j.cagd.2005.06.006]

Shoemake, K., 1985. Animating rotation with quaternion curves. *Computer Graphics (SIGGRAPH'85)*, **19**(3):245-254. [doi:10.1145/325165.325242]

Wallner, J., Dyn, N., 2005. Convergence and C^1 analysis of subdivision schemes on manifolds by proximity. *Computer Aided Geometric Design*, **22**(7):593-622. [doi:10.1016/j.cagd.2005.06.003]

Zefran, M., Kuman, V., Croke, C.B., 1998. On the generation of smooth three-dimensional rigid body motions. *IEEE Transactions on Robotics and Automation*, **14**(4):576-589. [doi:10.1109/70.704225]

Zefran, M., Kumar, V., Croke, C.B., 1999. Metrics and connections for rigid-body kinematics. *The International Journal of Robotics Research*, **18**(242):1-16.