Steady Affine Motions and Morphs

Jarek Rossignac
School of Interactive Computing
College of Computing
Georgia Institute of Technology
and
Àlvar Vinacua
Department of Software
Technical University of Catalonia

We propose to measure the quality of an affine motion by its steadiness, which we formulate as the inverse of its Average Relative Acceleration (ARA). Steady affine motions, for which ARA$=0$, include translations, rotations, screws, and the golden spiral. To facilitate the design of pleasing in-betweening motions that interpolate between an initial and a final pose (affine transformation), B and C, we propose the Steady Affine Morph (SAM), defined as $A^t \circ B$ with $A = C \circ B^{-1}$. A SAM is affine-invariant and reversible. It preserves isometries (i.e., rigidity), similarities, and volume. Its velocity field is stationary both in the global and the local (moving) frames. Given a copy count, n, the series of uniformly sampled poses, $A^{\frac{i}{n}} \circ B$, of a SAM form a regular pattern, which may be easily controlled by changing B, C, or n, and where consecutive poses are related by the same affinity $A^{\frac{1}{n}}$. Although a real matrix A^t does not always exist, we show that it does for a convex and large subset of orientation-preserving affinities A. Our fast and accurate Extraction of Affinity Roots (EAR) algorithm computes A^t, when it exists, using closed form expressions in two or in three dimensions. We discuss SAM applications to pattern design and animation and to key-frame interpolation.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Hierarchy and geometric transformations*

General Terms: Interpolating motions, Affinities

Additional Key Words and Phrases: Regular Patterns

This work was supported in part by CPA-G&V-T grant number 0811485 from the National Science Foundation, grant TIN2010-20590-C02-01 from the "Ministerio de Ciencia e Innovación" of the Spanish Government, and grant 2008 BE1 00018 of the Generalitat of Catalonia.
Author's addresses: J. Rossignac email: `rossignac@gatech.edu`
À. Vinacua email: `alvar@lsi.upc.edu`

DOI 10.1145/XXXXXXX.YYYYYYY
http://doi.acm.org/10.1145/XXXXXXX.YYYYYYY

1. INTRODUCTION

In this paper, the term "affine motion" (or simply "motion") refers to a continuous map from time t to an affinity A_t, which defines a pose (moving frame). Hence a motion transforms space by a composition of time-varying translations, rotations, uniform scalings, and shears. A more general (non-affine) animation of a deforming shape may be decomposed into an affine (carrier) motion, which can be extracted through least square registration [Bregler et al. 2002], and the residual free-form deformation.

Animation systems support tools for creating and editing in-betweening motions that interpolate a series of key-frame poses. There is an infinity of affine morphs (in-betweening motions) that interpolate a pair of initial and final poses, B and C, such that, with the proper choice of the time interval, $A_0 = B$ and $A_1 = C$. Which one is the most beautiful? The answer may depend on the context and on subjective criteria, which usually do not have a precise mathematical formulation.

Instead of measuring beauty, which is subjective, we propose to measure steadiness, which we define as the inverse of the Average Relative Acceleration (ARA). We formulate ARA as the average magnitude (over space and time) of the acceleration vector by which the relative velocity expressed in the moving frame changes over time.

Steady motions, for which ARA=0, include constant velocity translations, constant angular velocity rotations around a fixed axis, screw motions, and also the famous golden spiral observed in various natural growth formations and in stellar motions.

In Section 2, we define the Steady Affine Morph (SAM) from pose B to pose C as $A^t \circ B$ with $A = C \circ B^{-1}$.

Given a copy count, n, the series of uniformly sampled poses, $A^{\frac{i}{n}} \circ B$, of a SAM form a regular pattern, which may be conveniently edited by changing B, C, or n, and where consecutive poses are related by the same affinity $A^{\frac{1}{n}}$ (Figure 1).

Fig. 1. Left: Original regular pattern from F_0 (green) to F_n (red). Center: New regular pattern for a different ending pose. Right: The same poses F_0 and F_n, but a different copy count n.

In Section 4 , we present our efficient and accurate Extraction of Affinity Roots (EAR) algorithm for computing A^t when it exists in two and in three dimensions. In the Appendix, we provide implementation details and the derivations of the closed formulae used by EAR.

In Section 5, we show that A^t exists, as a real affinity, for a convex and comfortably large subset of orientation-preserving affinities A in two and in three dimensions. In Section 5.3, we propose efficient constructions of smooth-looking, piecewise-steady morphs for the rare configurations where SAM does not exist.

In Section 6, we prove several useful (and sometimes surprisingly beautiful) properties of SAMs. For example, a SAM is in general unique, affine-invariant, and reversible. Its velocity field is stationary both in the global and in the moving frame. It preserves rigidity (when A is an isometry, the SAM is a screw), similarity (when A is a similarity in 3D, the SAM is the composition of a 2D logarithmic spiral motion with an exponential scaling in the third dimension), and volume in 3D or area in 2D (when A does).

In Section 3, we discuss prior art and show that (except for cases where A is an isometry or a 2D similarity) previously proposed solutions are not steady. In Sections 2 and 6.4, and in the accompanying videos posted on the ACM website, we show that non-steady alternatives exhibit undesirable accelerations and non-monotonic changes of volume or area.

In Section 7, we discuss potential applications of SAM to collision detection, to the computation of envelopes of swept regions, to the design of regular patterns and of their animation, and to the creation of smooth-looking, piecewise steady motions that interpolate a series of key-frame control-poses.

2. PRELIMINARIES

In this section, we introduce our terminology and notation, define our solution, compare it to others, and discuss its properties.

2.1 Affinities

Here, we remind the reader of a few elementary properties of affinities. More detailed discussion can be found in many books, for example [Agoston 2005].

An affinity is a homeomorphism between $\mathbb{R}^n$ and itself and hence is invertible. It preserves convexity, co-linearity, parallelism, and ratios along a line, but in general, does not preserve angles or distances.

A transformation is affine when the image $A(\mathbf{p})$ of an arbitrary point $\mathbf{p} = (x, y, z)$ can be expressed as $\mathbf{t} + x\vec{\mathbf{i}} + y\vec{\mathbf{j}} + z\vec{\mathbf{k}}$ for some fixed set of linearly independent vectors $\vec{\mathbf{i}}$, $\vec{\mathbf{j}}$, and $\vec{\mathbf{k}}$ and for some point $\mathbf{t}$. For an affinity A, the vectors $\vec{\mathbf{i}}, \vec{\mathbf{j}}$ and $\vec{\mathbf{k}}$ are the images by A of the vectors of the universal orthogonal basis $\{(1,0,0),(0,1,0),(0,0,1)\}$, while $\mathbf{t}$ is the image by A of the origin $\mathbf{o} = (0,0,0)$. Hence, we represent A by the quadruple $[\vec{\mathbf{i}}\,\vec{\mathbf{j}}\,\vec{\mathbf{k}} \,|\, \mathbf{t}]$, for clarity using a vertical bar to separate the linear part (vectors) from the translation (point). A defines a local frame, which we call pose. It can be stored as a 3×4 matrix. For consistency with homogeneous matrix packages, A may also be stored as a 4×4 matrix. Such matrices are used in constructive representations of solid models, scene graphs, and animations. Their compositions are supported by graphic adapters.

We use the symbol L to denote the linear part of A, i.e., the 3×3 sub-matrix $[\vec{\mathbf{i}}\,\vec{\mathbf{j}}\,\vec{\mathbf{k}}]$. We can write $A(\mathbf{p}) = L(\overrightarrow{\mathbf{op}}) + \mathbf{t}$, which displaces the point $\mathbf{t}$ by vector $L(\overrightarrow{\mathbf{op}})$.

An affinity may be specified in various ways.

For example, it may be defined as the composition of a series of parameterized primitive transformations (rotation, translation, scaling, shear) [Mortenson 2007], which are supported by most design systems.

An affinity may also be specified as the *difference* (more precisely as the composition of one with the inverse of the other) between affinities B and C. For example, consider a shape F and its two instances, $B(F)$ and $C(F)$. We want an affinity A that transforms $B(F)$ into $C(F) = A(B(F))$. Hence, by the associativity of matrix multiplication $C = A \circ B$ and $A = C \circ B^{-1}$.

Finally, an affinity in $\mathbb{R}^n$ may be specified by $n + 1$ pairs of matching points $(\mathbf{s}_i, \mathbf{f}_i)$, such that both $\{\mathbf{s}_i\}$ and $\{\mathbf{f}_i\}$ are linearly independent sets. These points define $n + 1$ point-equality constraints: $\mathbf{f}_i = A(\mathbf{s}_i)$ for i in $\{0, 1, \ldots, n\}$, each resulting in n equations, one per coordinate. One may compute the coefficients of A by solving the corresponding system of $n(n + 1)$ linear equations. Notice that the solution is quite simple, since the system decouples into blocks of $n + 1$ rows and columns, and one needs only invert that sub-matrix and apply the inverse to the different coordinates. Alternatively, we may define affinities B and C that map the standard basis onto the frames defined by the $\{\mathbf{s}_i\}$ and the $\{\mathbf{f}_i\}$ respectively (for instance $B = [\overrightarrow{\mathbf{s}_0\mathbf{s}_1}\,\overrightarrow{\mathbf{s}_0\mathbf{s}_2}\,\overrightarrow{\mathbf{s}_0\mathbf{s}_3} \,|\, \mathbf{s}_0]$), and compute $A = C \circ B^{-1}$, as justified above. When more than $n + 1$ point-pairs are given, least-squares registration [Bregler et al. 2002] may be invoked to compute an optimal fit.

Here, we focus on orientation-preserving affinities in two and three dimensions. Therefore, the determinant of L must be strictly positive. Affinities for which the determinant of L is not positive cannot be animated by a SAM, because any continuous path joining an orientation-preserving affinity and another involving a symmetry must contain a singular transformation (one with zero determinant), which is not an affinity.

2.2 Steady Affine Morph (SAM)

An affine motion, not to be confused with a transformation, is a continuous mapping from the real line (time values) to the set of affinities. Let A_t denotes the affinity associated with time value t.

If we are given two poses (affinities), B and C, and wish to compute a motion that progressively interpolates between them, we use, as explained above, $A = C \circ B^{-1}$ and to each time t associate the affinity $A_t \circ B$, thus applying A_t in the frame of B.

Without loss of generality, we can shift and scale time so that the motion starts at time 0 (and hence A_0 is the identity $\mathbb{1}$) and so that, at time $t = 1$, we obtain the desired affinity A (hence $A_1 = A$).

Note that an $\mathbb{1}$ and A pair (or equivalently a B and C pair) does not completely define the interpolation, since an infinity of affine motions exist that satisfy these constraints.

To remove this ambiguity, we require (when possible) that the motion be steady. This term is precisely defined:

> An affine motion A_t is steady if and only if there exists an affinity A such that
>
> $$A_t = A^t \qquad (1)$$

We believe that this definition is novel and important because it provides a canonical form of an affine interpolating motion that happens to be aesthetically pleasing and mathematically beautiful.

Hence, we define the Steady Affine Morph (abbreviated SAM) from starting pose B to ending pose C as the steady interpolating motion $A^t \circ B$ with $A = C \circ B^{-1}$.

Although elegant, Equation (1) uses a non-standard notation of an affinity raised to a real power t. The equivalent notation

$$A^t = e^{t \log(A)} \tag{2}$$

is more standard. However, it involves the log of a 3×4 matrix. (One could consider computing the log of the corresponding homogeneous 4×4 matrix, but, as far as we know, there is no simple closed form expression for extracting the logarithm of a 4×4 matrix, whereas we provide such closed forms to compute a meaningful A^t.)

The notion of using the exponential to parameterize a group of transformations dates back to the seminal work of Sophus Lie in the nineteenth century. When restricted to the case of rotations, it has been advocated for graphical applications by Grassia [Grassia 1998]. Alexa [Alexa 2002] has also used it to compute linear combinations of homogeneous matrices. The concept of steadiness introduced here differs from the "As-Rigid-as-possible" property introduced in [Alexa et al. 2000] in three ways:

(1) Steadiness has a precise mathematical definition,
(2) Steadiness defines the full affine motion, not just its linear part, and
(3) A steady motion that interpolates two poses is unique.

2.3 SAM properties

SAMs have several remarkable and useful properties:

(1) When it exists, the SAM that interpolates two given poses is unique (discarding SAMs that rotate by more than 360 degrees and excluding 180 degrees turns).
(2) The SAM that interpolates two poses is affine-invariant (i.e., it is not affected by a change of coordinate system). The importance of this property was stressed in [Rossignac and Kim 2001; Lee and Shin 2002; Kavan et al. 2008].
(3) A SAM is reversible (i.e., direction-independent): the reverse motion is obtained by swapping the first and last poses. The importance of this property was claimed in [Kavan et al. 2008].
(4) Integral measures (volumes, area) evolve monotonically during a SAM (see Section 6.4), which implies that they remain constant if A preserves them.
(5) When interpolating rigid transformations (isometries) in 3D, the SAM is a screw and hence preserves rigidity (which is often desired in animation [Alexa et al. 2000]) and satisfies the shortest path (smallest rotation angle) and constant speed (linear variation of rotation angle and translation distance) properties advocated in [Kavan et al. 2008].
(6) During a SAM, the velocity of every point remains constant in both the local and the global frame. This property may be verified in Figure 2. It simplifies the computation of the envelope of the swept region (see [Rossignac et al. 2007]). We show in Section 6.1 that SAMs are the only motions with this property.
(7) The motion followed by a point during a 3D SAM combines a logarithmic spiral in a plane with an exponential scaling in a direction transversal to the plane. The simplicity of the mathematical description of this motion facilitates the computation of point/plane collisions, hence suggesting possible extensions of the collision prediction techniques, originally developed for screw motions [Kim and Rossignac 2003], to affine motions.
(8) SAM produces visually pleasing animations, where points travel along naturally arcing trajectories without unexpected inflections. Animation artists often prefer such trajectories to linear motions [Johnston and Thomas 1995; Whited et al. 2010]. For example, in Figure 2, notice how SAM (right column) achieves the most uniformly spaced and least distorted pattern of pictures.

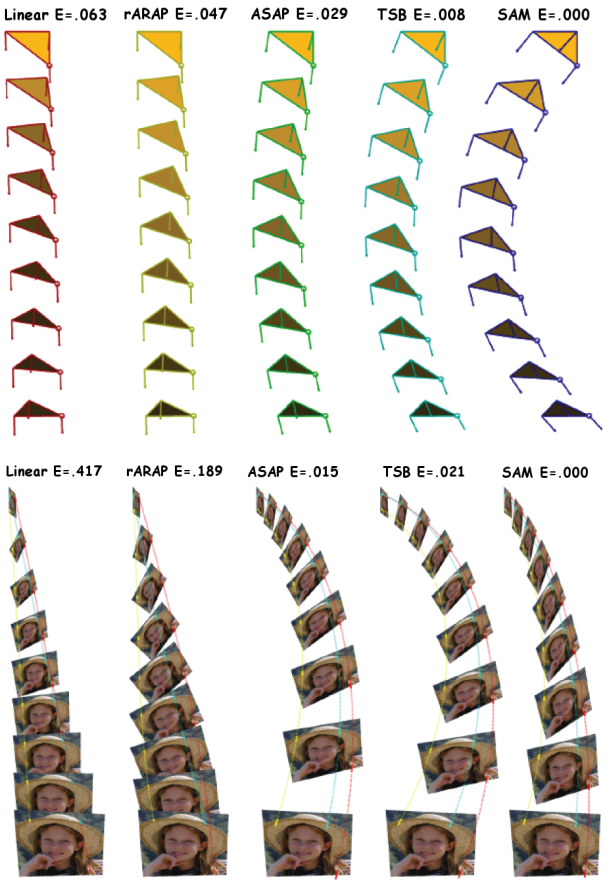

Fig. 2. A comparison of different motions acting on a triangle (top) and on a textured quad (bottom). Notice also that, during a SAM (right column), the velocity vectors drawn at the triangle vertices remain constant, in the local frame. For example, the velocity of the top vertex of the triangle always ends at the same relative place near the opposite edge.

To illustrate, in a static picture, the fact that SAM evolves smoothly as the user changes the starting and/or the ending pose, we show in Figure 3 a slowly changing series (left-to-right) of starting (top) and ending (bottom) pose-pairs (produced themselves as intermediate poses of SAMs) and, for each pair, we show the interpolating SAM as a vertical row (regular pattern) of shapes.

2.4 Comparison with other morphs

Here, we contrast SAM with other (unsteady) morphs using a two-dimensional example. Figure 2 compares five in-betweening motions that interpolate the same pair of starting and ending (control) poses using a triangle (top) and a rectangular image (bottom).

The columns correspond (left to right) to (a) linear interpolation of vertex positions, (b) a modified ARAP [Alexa et al. 2000], where a constant-speed translation along the line joining the barycenters has been added (because ARAP only deals with the evolution of the shape and its orientation, but not with its position), (c) a loga-

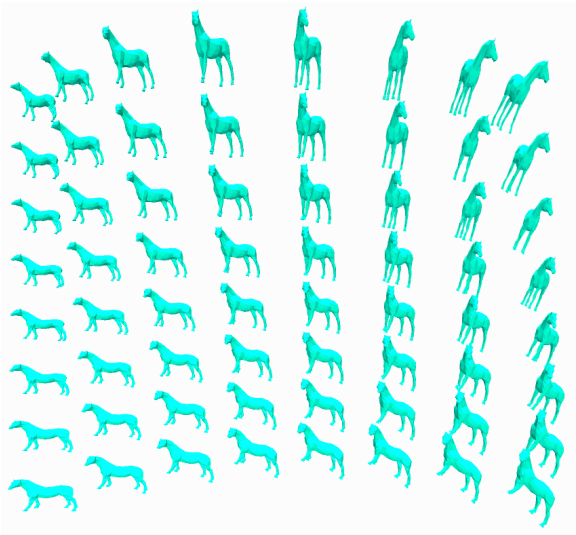

Fig. 3. A series of SAMs, each one shown as a vertical row of poses, for a smoothly varying (left-to-right) set of starting and ending poses.

rithmic spiral motion, which interpolates the positions, orientations and uniform scaling, combined with a linear interpolation of the residue, (d) a blending (weighted average) of three logarithmic spirals computed from the three pairs of sides of the triangles (as used in [Whited et al. 2010]), and (e) the SAM proposed in this paper.

For each morph, we report our ARA measure of unsteadiness. We formulate this measure as the integral over space and time of the acceleration in the moving frame. We compute it using a dense regular sampling of a vicinity of the moving shape and obtain, Top: linear=.063, rARAP=.047, ASAP=.029, TSB=.008, and SAM=.000. Bottom: linear=.417, rARAP=.189, ASAP=.015, TSB=.021, and SAM=.000.

3. PRIOR ART

We split prior art into three groups: (1) the special case of screw motions, (2) previously proposed numeric solutions for extracting roots and logarithms of linear transformations, and (3) unsteady morphs.

3.1 Steady interpolation of isometries

Since we assume that L (the linear part of affinity A) is orientation preserving, when the column vectors of L form an orthonormal basis, A represents an isometry (i.e., a rigid body transformation) and may be expressed as a rotation (defined by the matrix L) and a translation (from the origin to point $\mathbf{t}$).

In two dimensions, the SAM of a rigid body transformation is either a constant velocity translation (when $L = \mathbb{1}$) or a rotation with constant angular velocity around a fixed point $\mathbf{q}$, which may be easily computed by solving the linear system of two equations $A(\mathbf{q}) = \mathbf{q}$. The angle a of the rotation is defined by the first column vector of L: $(\cos(a), \sin(a))^\mathsf{T}$.

In three dimensions, the corresponding SAM is a screw motion [Rossignac and Kim 2001], which combines a constant angular velocity rotation (by angle ta) around a fixed axis (with direction $\vec{\mathbf{d}}$ and passing through the fixed point $\mathbf{q}$) and a constant velocity translation (by distance td) in the direction $\vec{\mathbf{d}}$.

Simple closed-form expressions have been proposed for computing the screw parameters ($\vec{\mathbf{d}}$, $\mathbf{q}$, a, and d) using variations of the Rodrigues formula [Rossignac and Kim 2001; Llamas et al. 2003]. The screw motion may also be computed and animated using dual quaternions [Clifford 1882], which offer benefits for GPU support and for computing weighted averages of more than two rigid body transformations. For example, the Screw Linear Interpolation (ScLERP) proposed in [Kavan et al. 2008] as a generalization of SLERP [Shoemake 1985] produces screw motions.

Note that, when applied to the special case of a rigid body motion, A, the more general EAR approach proposed here computes the correct $\vec{\mathbf{d}}$ as the eigenvector of the unique real eigenvalue of L and $\mathbf{q}$ as the fixed point of A. (The SAM of a rigid transformation A is the corresponding screw motion.)

3.2 Numeric solutions for computing $\log(L)$

We discuss here previously proposed iterative approaches for computing $\log(L)$, which is the fundamental challenge to be solved when computing A^t in the general case, see Equation (9).

There is abundant bibliography on the computation of exponentials, logarithms and square roots of matrices of arbitrary dimension. See for example [Meini 2005; Cheng et al. 2000; Golub and Van Loan 1996; Davies and Higham 2003] or [Higham 1986]. The reported algorithms are also available on standard platforms, such as Matlab [Higham and Higham 2005].

A approach based on the Schur decomposition [Davies and Higham 2003] performs a change of basis to obtain an upper-triangular matrix. This decomposition is known (see for example [Golub and Van Loan 1996]) to be cheaper to compute than the Jordan normal form, and to be also more stable numerically. Since the change of basis commutes with the logarithm and with exponentiation, the problem is reduced to the simpler problem of computing the log of an upper-triangular matrix. This seems the best approach for matrices of arbitrary dimension, but is an unnecessary complication compared to our straightforward formulas for fixed dimension 2 or 3.

As pointed out in [Kavan et al. 2008] these general, dimension-independent, solutions suffer from numeric errors. Furthermore, their robust implementation is delicate because of possible convergence and branching problems. For example, consider the two dimensional matrix $A = \begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix}$ which has a double negative eigenvalue. The `logm` functions of both Matlab and Octave —a public domain clone of Matlab (see [Eaton 2000])— return a complex solution, when in fact a real solution, $\begin{bmatrix} \log(2) & -\pi \\ \pi & \log(2) \end{bmatrix}$, exists and is correctly computed by our EAR implementation.

The closed-form solution proposed here for computing —when it exists— the real log of 2×2 and 3×3 matrices, may help address the issues of computational cost and numerical stability discussed above.

The discussion of the existence of a real solution is deferred to Section 5.

3.3 Unsteady interpolations

In this subsection, we discuss previously proposed interpolation techniques that produce unsteady motions. We also discuss the concept of rigidity introduced in prior art and its relation to the concept of steadiness introduced here.

We split the unsteady interpolation techniques into three categories: (1) Those dealing with rigid body transformations. (2)

Those dealing only with the linear sub-matrix L, hence ignoring the translation part of the motion (such an approach is appropriate when only the shape and orientation of the moving object are important or when its center of mass must follow a trajectory prescribed by physics or artistic concerns). And (3) those dealing with more general affinities.

3.3.1 *Rigid body transformations.* Several authors have proposed techniques for blending spherical spline curves on the S^3 unit sphere [Buss and Fillmore 2001b; Shoemake 1985; 1987; Duff 1986; Wang and Joe 1993; Roberts et al. 1988; Kim and Nam 1995]. Barr et al. [1992] use quaternions, but compute a smooth rotation that interpolates a given sequence of constraint poses. Kim et al. [1995] generalize this approach by providing a general framework for interpolating motions with unit quaternion splines. Ma et al. [2000] used B-splines to compute smooth interpolating rigid body motions. They first approximate the reference frames by B-splines that do not preserve orthogonality, and then compute the best orthogonal approximation through a Newton iteration (for each frame).

Kavan et al. [2008] compare their (ScLERP) approach to two other unsteady solutions: DLB and DIB. Their Dual quaternion Linear Blending (DLB) is a generalization of the QLB [Kavan and Žára 2005], which performs a linear interpolation of quaternions, followed by a normalization. DLB uses the same fixed screw axis as ScLERP, but changes the rotation angle and translation distance non-linearly and differently. The discrepancy is small. The Dual quaternion Iterative Blending (DIB) is a generalization of spherical averages [Buss and Fillmore 2001a].

3.3.2 *Interpolations of L.* Techniques described in this subsection focus only on the linear part, L, of the transformation, hence ignore translation.

The linear interpolation of the coefficients of L is often not acceptable because it may temporarily invert the shape, and may evolve the area/volume in a non-monotonic fashion.

Shoemake [1985] and Shoemake and Duff [1992] use spherical linear interpolation (SLERP) to interpolate orientations represented using quaternions and combine it with a linear interpolation of the coefficients of a stretch matrix. In [Shoemake and Duff 1992], they use the polar decomposition to parameterize and animate an affinity. They decompose an affinity A into the product $R \circ S$ of a rigid motion R and a symmetric positive-definite stretch matrix S. R is computed by minimizing the Frobenius norm $||A - R||$. Then, both R and S are parameterized by time and the animated affinity composed as $R(t) \circ S(t)$. $R(t)$ is animated as a screw and $S(t)$ is animated using a linear interpolation of the matrix coefficients. Shoemake and Duff apply their approach to construct smooth motions that interpolate more than two affine transformations. Computing independent splines that interpolate the coefficients of the matrices of these transformations may produce unwanted distortions because the orthogonality of $R(t)$ and the rigidity of $S(t)$ are not enforced.

Alexa et al. (see [Alexa et al. 2000]) use a decomposition similar to the polar decomposition in [Shoemake and Duff 1992], but instead of finding the rotation through a minimization, they use the SVD (Singular Value Decomposition, see [Golub and Van Loan 1996]) of L. The SVD returns two orthogonal matrices U and V, and a diagonal matrix Y, such that $L = U^{\mathsf{T}}YV$. These orthogonal matrices are pure rotations when their determinant is positive (i.e. when it is 1). SVD may return matrices with negative determinants, but in this case, and if the affinity is orientation-preserving, exchanging the variables will yield orthogonal matrices with positive determinant, from which we can then extract the angles (α and β). The linear part of the affinity can then be written as $L = R(\alpha) \circ R(\beta) \circ (R(-\beta) \circ Y \circ R(\beta)) = R(\alpha+\beta) \circ B$, where $B = R(-\beta) \circ Y \circ R(\beta)$ is a positive definite stretch matrix, analogous to (but different from) S in [Shoemake and Duff 1992]. Then, a morph of L may be formulated as $L(t) = R(t(\alpha+\beta)) \circ ((1-t) \cdot I + t \cdot B)$. Notice that since α and β are computed independently, their sum's absolute value may exceed 2π, in which case the authors of ARAP recommend adding/subtracting 2π (see also [Fu et al. 2005; Choi and Szymczak 2003] for a more detailed discussion).

Furthermore, note that the decomposition proposed in [Alexa et al. 2000] produces, in general, a different solution than the just-as-valid, decomposition: $L(t) = ((1-t) \cdot I + t \cdot B) \circ R(t(\alpha+\beta))$, with $B = R(\alpha) \circ Y \circ R(-\alpha)$. This possibility of choosing between two different solutions suggests that neither one is optimal and that an optimal solution exists that is "more rigid" than the "As-Rigid-As-Possible" solution of [Alexa et al. 2000].

We have attempted to produce a steady solution by using our ASAP variation of the ARAP scheme, obtained by replacing the linear interpolation between the identity and B by an exponential interpolation. This modified solution would yield: $L(t) = R(t(\alpha+\beta)) \circ R(-\beta) \circ Y^t \circ R(\beta)$. Unfortunately, this improved solution is still not steady, and in fact produces a different result from the similar, and equally valid variation: $L(t) = R(\alpha) \circ Y^t \circ R(-\alpha) \circ R(t(\alpha+\beta))$.

Alexa [2002] approximates $\log(L)$ by a series expansion. Since such an approach only converges for matrices close to the identity, Alexa first computes an approximation of L^t by performing a series of square root extractions during a binary partition of the unit time interval. Because of the binary search and the series expansion, this approach is slower than our EAR solution. Furthermore, to extract the square roots, Alexa relies on the Denman-Beavers iteration, which was found to be unstable [Higham 1986]. In fact, our equation (2) is consistent with Equation (7) in [Alexa 2002], where the linear combination $(1-t)\mathbb{1} + tA$ is expressed as $e^{(1-t)\log(\mathbb{1})+t\log(A)}$, which simplifies to $e^{t\log(A)}$, since $\log(\mathbb{1}) = [0]$. Yet, our solution differs from Alexa's in two important ways:

(1) When interpolating between B and C, Alexa proposes to use $e^{(1-t)\log(B)+t\log(C)}$ which is $e^{\log(B)+t(\log(C)-\log(B))}$ and therefore is not equivalent to the expression we use, namely $e^{t\log(C\circ B^{-1})} \circ B$, except in singular cases when B and C commute. Hence, when B and C do not commute (which is the general case), the solution proposed in [Alexa 2002] is not steady. For example, consider the 2D case where $B = \mathrm{Rotate}(90^\circ)$, and $C = \mathrm{Translate}(6, 0)$. Extract the mid-course transformation $X = A_{1/2}$. Our SAM formulation yields

$$X_{SAM} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 3 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 3\sqrt{2}-3 \end{bmatrix}.$$

Note that $X_{SAM}^2 \circ B = C$, and hence $X_{SAM} = A^{1/2}$, which is consistent with the definition of a steady motion. In contrast, the formulation proposed in [Alexa 2002] yields a different result:

$$X_{Alexa} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{6\sqrt{2}}{\pi} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -\frac{6\left(\sqrt{2}-2\right)}{\pi} \end{bmatrix}$$

showing that the motions produced by the scheme of [Alexa 2002] are not steady; one may observe that, in this simple case, both solutions turn the model by the same amount, but $X_{SAM}^2 \circ$

$B = C$, while $X^2_{Alexa} \circ B$ is a translation by only $\frac{12\sqrt{2}}{\pi} \approx 5.402$ in the x direction, and hence is different from C (see Figure 4). Notice that when B and C commute (for example if $B = \mathbb{1}$), the two solutions are identical.

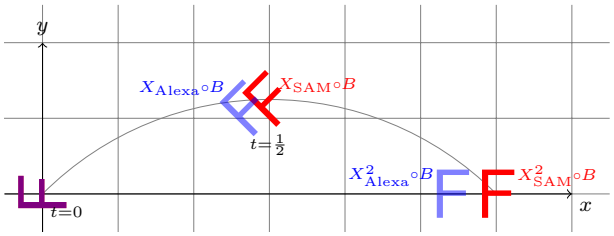

Fig. 4. The mid-way pose produced by SAM (red) amounts to a rotation by $-\frac{\pi}{2}$ around the fixed point $\mathbf{q} = (3, -3)$. The mid-way pose produced using the interpolation of [Alexa 2002] is shown in blue. Squaring the corresponding transformation does not interpolate the final position, showing that the scheme is not steady.

(2) Alexa computes $\log(A)$ for a 4×4 matrix A using dimension independent, iterative, numeric tools. Instead, we propose here a new approach that is limited to two and three dimensions, but performs the computation using closed form expressions directly, without iterations. This new solution has robustness and performance benefits.

We provide a comparison (Figure 5) of SAM (red) with the exponential interpolation proposed in [Alexa 2002] (blue). By inspecting the overlaps between pairs of consecutive triangles, notice how the blue triangle moves and rotates more slowly at the left, and more quickly at the right, whereas the constant velocity of the red triangle makes consecutive poses overlap in the same way.

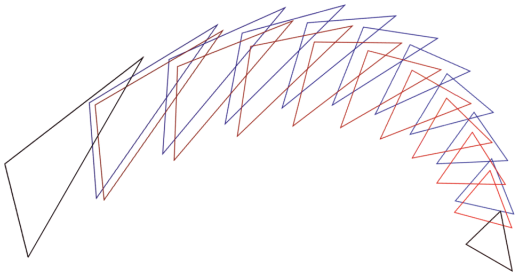

Fig. 5. The motions defined by SAM (red) and by the algorithm of [Alexa 2002] (blue) between the two triangles drawn in black, sampled at equal intervals in time.

In their Dual quaternion Iterative Blending (DIB), Kavan et al. [2006] propose an iterative extension of Alexa's 'log-matrix blending' ([Alexa 2002]) that makes a constant-speed, shortest path solution, but warn that the matrix exponential and logarithm routines it uses require numeric solutions to be applied iteratively, which reduces performance and accumulates numeric errors.

3.3.3 *Steadiness and rigidity.* In their "As-Rigid-As-Possible" (ARAP) approach, Alexa et al. [2000] refer to the rigidity of an interpolating motion, but do not offer a formal definition of rigidity. In contrast, steady motions are precisely defined here, and ensure that the shape deformation is as rigid as possible, and at the same time ensure that the overall motion is free of undesired undulation.

Hyun et al. [2002] propose an approach for progressively refining an affine spline motion using an iterative procedure that involves knot insertion and degree elevation and operates on a curve in a linear 12-dimensional space (of the affine matrix coefficients), where they relate the fairness of the curve in 12-space to the rigidity and fairness of the motion in 3-space. Hence, their rigidity measure corresponds to non-quadratic fairness measures and requires a numerical approximation. They measure rigidity by integrating over time the first or second derivatives of the lengths of a certain set of witness vectors as they are transformed (see their equations (12) and (13)). Their measure differs from the ARA measure proposed here. Because of the non-linear nature of the problem, their solution requires an iterative minimization. In cases where the final pose is the image under a rigid motion of the original, the corresponding SAM will make all the witness vectors constant in length, and therefore both measures will be minimized (and will actually be zero). However, in other more general cases, the measure proposed in [Hyun et al. 2002] will depend on the choice of witness vectors. Changing those will likely yield a different solution. EAR yields a unique solution, the steady one, and therefore is different from the solution proposed in [Hyun et al. 2002].

4. EAR (OVERVIEW)

The closed-form solution proposed here for computing A^t in 3D, when it exists as a real matrix, distinguishes: (a) three types of general configurations, which correspond to full-dimensional subsets of the twelve-dimensional space of affinities, and (b) several singular configurations, which correspond to the lower-dimensional boundaries of the general configuration subsets, and represent special cases, such as a simple translation or a rotation by 180 degrees.

The various special cases of the singular configurations can be identified reliably using simple closed-form Boolean expressions and are each treated using a specific formulation for A^t. Although these special cases increase the complexity of the implementation and of its description, we chose to treat these special cases explicitly (rather than to approximate them by a nearby general configuration obtained by a small perturbation) because exact solutions may be required in mechanical or architectural applications.

Our approach for identifying which type of general configuration corresponds to a given affinity A and for producing the closed form construction for A^t is based on two observations.

(1) We can express $A^t(\mathbf{p})$ as $L^t(\overrightarrow{\mathbf{qp}}) + \mathbf{q} = e^{tlog(L)}(\overrightarrow{\mathbf{qp}}) + \mathbf{q}$, where $\mathbf{q}$ is a fixed point (which exists and is unique in general configurations).

(2) In 3D, L must have at least one real positive eigenvalue. Therefore, L may be expressed as the commutative product of a scaling along the corresponding eigenvector $\overrightarrow{e_1}$ and of a 2D linear transformation. Hence, we compute L^t as the product of an exponential scaling along $\overrightarrow{e_1}$ and of a 2D SAM in the linear sub-space spanned by the other two —probably generalized— eigenvectors.

Assuming that the singular cases have been screened and that we are hence in a general case, to solve the 2D version of our problem, we distinguish three types of general configurations, based on the characteristic polynomial of a 2×2 matrix L (which is the linear

part of A in 2D or the restriction of A to the appropriate subspace in 3D):

(1) L has two real positive eigenvalues: This configuration corresponds to the simple case where L can be diagonalized.
(2) L has two complex conjugate eigenvalues: We provide a closed form solution that does not use complex numbers.
(3) L has two real negative eigenvalues: This configuration corresponds to cases where no real solution exists, and hence there is no SAM. We discuss in Section 5 unsteady or piecewise-steady solutions for such bad configurations.

Note that real eigenvalues of L cannot have opposite signs, since L is orientation preserving.

In the Appendix, we provide and justify the derivation of our closed form expressions for identifying the appropriate general or singular case and for computing the corresponding A^t in two and three dimensions, when it exists.

5. EXISTENCE OF A SAM

Considering that a real $\log(A)$ may not exist, we address here several important questions:

(1) Does EAR always produce a solution when one exists?
(2) Is there a safe set of good configurations (for which SAM exists) that is sufficiently large to be useful for common applications?
(3) What unsteady motions should be used in the bad configurations?

5.1 Characterization of bad configurations

To compute the SAM, we need to compute L^t, which, depending on the case, can be done either through diagonalization or by computing $\log(L)$. Even when it is done through diagonalization, the conditions that L needs to meet (it diagonalizes into a matrix with positive entries) ensure the existence of a real $\log(L)$. Hence SAM exists when and only when $\log(L)$ is real. A precise characterization of when this happens is given in [Culver 1966]: the algebraic and geometrical multiplicities of each real negative eigenvalue of L must be equal and even. Notice that (contrary to what is stated in [Alexa et al. 2000]) an arbitrary orientation-preserving affinity does not necessarily satisfy this condition. However, all orientation preserving affinities with a homogeneous scaling (similarities) do have a real logarithm. Since we are interested here in dimensions 2 and 3, Culver's characterization means that

> For L to have a real logarithm, if there is a negative eigenvalue, it must be a double eigenvalue with two linearly independent eigenvectors. (3)

In the singular case of a double negative eigenvalue with two linearly independent eigenvectors, the affinity is a rotation by 180 degrees (possibly combined with a uniform scaling). Then both eigenvalues are $-s$, where s is the scale factor. However, general case affinities arbitrarily "near" this one may have distinct negative eigenvalues, and hence may not have a real logarithm.

Ignoring the singular configurations, an orientation-preserving 2×2 matrix L does not have a real log when its two eigenvalues, λ_1 and λ_2, are negative. This happens only when $d_L \geq 0$ and $\mathrm{trace}(L) \leq -2\sqrt{d_L}$, where $\mathrm{trace}(L)$ is the sum of the diagonal elements of L, and d_L is the discriminant of its characteristic polynomial, i.e. $d_L = \left(\frac{\mathrm{trace}(L)}{2}\right)^2 - \det(L)$.

5.2 Geometry of configurations where SAM exists

Culver's existence condition quoted in the previous Subection indicates that solvability (i.e., the existence of SAM) is not affected by changes in translation and in uniform scaling.

Furthermore, a three dimensional matrix L always has one "good" direction along which it is a scaling by a real eigenvalue. We may further always choose a positive eigenvalue, since we are considering orientation-preserving matrices, which cannot have, in three dimensions, all of their real eigenvalues negative. Therefore, solvability is determined by the restriction of L to the sub-space of the remaining generalized eigenvectors.

Hence, we need only discuss solvability in the space of all 2×2 matrices of determinant one, which can be constructed as $R(a) * R(-c) * Y(1/s, s) * R(c)$, where $R(x)$ denotes rotations around the origin by the given angle x, $Y(x, y)$ is a diagonal matrix with entries x and y, and where s and $1/s$ are the two singular values of L. Solvability is not affected by changing the angle c. Therefore, we only examine the solvability for (a, s) pairs, where unsolvable regions are painted red. Note that we only need to study solvability for $0 < s \leq 1$, since when $s > 1$ the other singular value $1/s$ lies between 0 and 1. Bad configurations, shown red in figure 6, satisfy $\mathrm{trace}(L) < -2\sqrt{d_L}$. Rewriting this inequality using the coefficients of $L = R(a) * Y(1/s, s)$ expressed in terms of a and s yields:

Fig. 6. Solvability for 2×2 matrices of determinant 1. Going from-left-to-right, the rotation angle a varies from $-\pi$ to π. Going up, the singular value s varies from 0 to 1, while $1/s$ varies from infinity to 1. The save region is convex and contains a trapezoid (black border).

$$|a| < cos^{-1}\frac{-2s}{s^2+1} \tag{4}$$

Observe that the good region, where SAM exists, is convex and contains a trapezoid defined by: $|a| < (1+s)\pi/2$.

For a different perspective on the size of the good region where SAM exists in three dimensions, we have checked a dense sampling of configuration parameters in a limited region in the space of 3×3 matrices. Since a uniform scaling does not affect the existence of a solution, our sample space consists of a set of directions, a set of rotations around those directions, and a set of perturbations of each element in the resulting matrix. To sample directions, we have constructed an icosahedron, subdivided it twice, and tested all axes emanating from the origin and passing through a vertex of the result. We then tested rotations around those axes from $-\theta$ to θ, and for each such rotation, we have added a perturbation to each element of the matrix L by $-p$, 0 or p in all possible combinations, and checked if the resulting matrix had a logarithm.

For each configuration, we report the total rotation angle θ and the perturbation p. When $\theta < 2\pi/3$ a SAM exists for all configurations where $p < 0.29$. The bound p decreases as the bound on the rotation angle is increased. Restricting $\theta < \pi/2$ allows for perturbations up to .33, while restricting $\theta < \pi/4$ allows for perturbations up to .50.

5.3 When no SAM exists

We have explored a variety of strategies for dealing with situations where no SAM exists.

The simplest strategy is to invite the designer (or use an unsteady motion) to provide an intermediate control pose. For instance, if no SAM exists between pose F_A and F_C, the designer may insert a new control pose F_B between F_A and F_C. The simplest way of using this new pose is to compute what we call a *polySAM*, a continuous motion concatenating a sequence of SAM spans: a SAM from F_A to F_B and a SAM from F_B to F_C. The burden is on the designer to adjust F_B so that both SAMs exist. Although this solution produces a continuous motion, it is only piecewise-steady and typically exhibits a sharp discontinuity of velocity when the moving shape passes the pose F_B (see Figure 7-b).

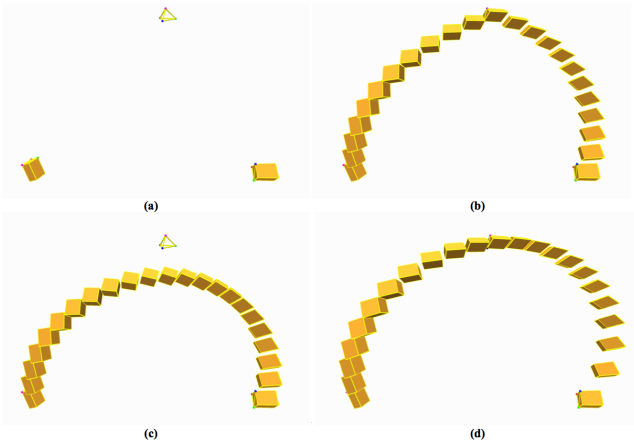

Fig. 7. The initial configuration (a) has no SAM between the F_A (lower left) and F_C (lower right) poses. The designer or an unsteady interpolation inserts an intermediate pose F_B (top center). A polySAM (b) that concatenates a SAM from F_A to F_B with a SAM from F_B to F_C shows a velocity discontinuity when passing at F_B. A quadratic Bézier motion (c) (i.e., a parabolic curve in the space of affinities) with control poses F_A, F_B, and F_C does not have such a discontinuity, but does not interpolate F_B. Our SAM bi-arc retrofit construction (d) produces a concatenation of two quadratic Bézier motions that interpolates F_B and is smooth.

To remove this discontinuity, if we do not require that the new motion interpolate F_B, but only that it pass close to it, we can use a motion defined by a quadratic Bézier motion (quadratic curve in the space of affinities) with control poses (F_A, F_B and F_C) and where SAM is used instead of linear interpolation (see Figure 7-c). We could also use a retrofitting procedure (similar to those discussed in [Rossignac and Schaefer 2008]) to compute a perturbation of F_B that brings the Bézier motion closer to the original F_B. Instead, we propose to retrofit a SAM bi-arc (Figure 7-d), a motion that concatenates two quadratic Bézier motions connected at F_B. This construction produces C^1 motions, although they are in general not steady. Velocity continuity follows from the observation that pose F_B is the mid-course pose of the SAM from F_L to F_R.

Our retrofit construction is very simple. Let `SAM(`F_A`, t, `F_B`)` return the pose at time t along a SAM that interpolates between pose F_A and pose F_B. Let `SAM(`F_X`, `F_A`, t, `F_B`)` return the image of pose F_X transformed by `SAM(`F_A`,t, `F_B`)`. Then, let `bezier(`F_A`, `F_B`, `F_C`, t)` return the affinity for time parameter t on a quadratic Bézier curve in the space of affinities with control poses (F_A, F_B, F_C), where a SAM is used as the primitive interpolation between two poses. We implement it as:

```
Pose bezier(Pose FA, Pose FB, Pose FC, float t)
 { return SAM(SAM(FA, t, FB), t, SAM(FB, t, FC)); }
```

We want to construct a motion that starts at F_A, passes at F_B exactly at $t = 1/2$, and ends at F_C. We concatenate two quadratic Bézier motions: one with control poses (F_A, F_L, F_B) and the other one with control pose (F_B, F_R, F_C). Specifically, when $t \leq 1/2$, we use a `bezier` with control poses (F_A, F_L, F_B) and parameter $2t$, and when $t > 1/2$, we use the symmetric construction:

```
if(t<=0.5) return bezier(FA, FL, FB, t*2);
   else return bezier(FB, FR, FC, (t-0.5)*2);
```

We compute F_L and F_R as $F_L =$ `leftControl`(F_A, F_B, F_C) and $F_R =$ `leftControl`(F_C, F_B, F_A), using

```
Pose leftControl (Pose FA, Pose FB, Pose FC)
     {return SAM(SAM(FA,0.75, FB), FC, .25, FB);}
```

Note that we have chosen a construction that only requires computing SAMs between F_A and F_B and between F_B and F_C, since the SAM between F_A and F_C may not exist.

In 2D, the automatic techniques that may be used to produce an unsteady motion serving to provide the intermediate pose F_B include our rARAP, ASAP, and TSB unsteady alternatives. In 3D, one could explore using extensions of these 2D solutions and also unsteady solutions previously proposed in the literature.

These solutions produce smooth, but unsteady motions. If the application can benefit from the properties of SAM, we simply produce a sequence of m poses uniformly sampled along the unsteady morph and produce a polySAM composed of a SAM between each pair of consecutive sample poses. The number m of samples may be used to fine-tune the proper compromise between velocity discontinuities at the inserted samples (which decrease as m grows) and processing cost (which increases with m as more SAMs must be processed).

6. PROPERTIES OF SAMS

Steady Affine Morphs have several beautiful properties. In this section, we prove a few, hoping to motivate further exploration.

6.1 Uniqueness

First, let us answer the following question. Given an affinity A, can there be more than one SAM that interpolates between the identity $\mathbb{1}$ and A?

Consider an affine motion A_t. Requiring that, at any time t, and for any point $\mathbf{p}$, the velocity at which $\mathbf{p}$ is moving in the local frame (the frame given by the columns of L) be constant over time, yields a system of ODEs:

$$\frac{d}{dt}L(t) = M \circ L(t), \tag{5}$$

and

$$\frac{d}{dt}\mathbf{t}(t) = L(t)\left(\begin{bmatrix}\sigma\\ \tau\end{bmatrix}\right) \tag{6}$$

where M is a constant matrix and σ and τ are constraints determined by the translation part $\mathbf{t}$. Without constraints, we may freely choose M, σ and τ. If however we want a certain initial simplex at time 0 to reach a specific position at time 1, then we need to solve for the appropriate values of M, σ and τ.

Note that the resulting motions are no other than SAMs. The solutions of Equation (5) are of the form $L(t) = e^{tM}$ [Arnold 1981], and since we want $L(1) = L$ (our time shifting and scaling convention), M must be $log(L)$. It then follows from the uniqueness of the solutions to these differential equations that there can only be one SAM between two given affinities if we fix the times (in our case 0 and 1) when the motion interpolates the given affinities. This holds except for branching points (a turn by 180 degrees may equally be interpolated by either increasingly positive or increasingly negative turns), and rotation cycles (when one reaches the destination after a rotation by more than 360 degrees), which we choose not to do. One easily checks by differentiation of the exponential that a SAM defined by Equation (1) or (2) satisfies this ODE and thus has constant velocity in the local frame. Therefore, the constant velocity condition enforced by (5) and (6) is an alternative, equivalent, definition of a SAM.

Notice also that by choosing the origin at the fixed point of the affinity A, the translation part is zero, and we therefore have $\sigma = \tau = 0$. This works in all cases safe some singular cases (when L has an eigenvalue 1), which are the only cases in which one needs to deal with these constants (see Sections A.5 and A.6). Even in those cases, appropriate choice of the origin of coordinates yields a vector $[\sigma, \tau]^\mathsf{T}$ which is an eigenvector of L of eigenvalue 1, and therefore for that reference $\frac{d}{dt}\mathbf{t}(t) = [\sigma, \tau]^\mathsf{T}$ is constant.

Furthermore, when L is a matrix that has no real logarithm, if there existed a constant velocity motion (in the local frame) $S(t)$ interpolating F_0 and $F_n = [L \,|\, \mathbf{t}](F_0)$, that motion would need to satisfy the previous differential equations. Then by the uniqueness of the solutions to the differential equation, we would have found the logarithm of L nonetheless, which would lead to a contradiction.

6.2 Constant velocity

Let us now investigate the instantaneous velocity of a point $\mathbf{p}$ subject to a SAM in 2D. In the local frame, according to the discussion in the previous subsection concerning equations (5) and (6), a SAM moves every point with constant velocity. In fact, in the local frame, the velocity vector of a point that starts at (x_0, y_0) at time $t = 0$ is given (in an appropriate basis as discussed in the previous section) by $M \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \sigma \\ \tau \end{bmatrix}$ and remains constant through time, since M and e^M commute.

Moreover, the velocity vector field is stationary (i.e. the velocity at each point in the plane does not change with time). Indeed, in the general case, where 1 is not an eigenvalue of L, our choice of origin makes $\mathbf{t}(t) \equiv 0$, and we see that the velocity at the point $\mathbf{r}$ is $M(\mathbf{r})$, by considering any trajectory that visits $\mathbf{r}$ at some given time t:

$$\mathbf{r} = \mathbf{p}(t) = A_t(\mathbf{p}) = L(t)(\mathbf{p}) = e^{tM}(\mathbf{p})$$

we see by differentiating that the velocity of $\mathbf{p}(t)$ is

$$\mathbf{p}'(t) = M \circ e^{tM}(\mathbf{p}) = M(\mathbf{p}(t)) = M(\mathbf{r}).$$

The fact that we have chosen a special origin that makes $\mathbf{t} \equiv 0$ does not restrict the conclusion and is only a convenience to make the computation simpler. In the cases where L does have some eigenvalue equal to one, as discussed in Sections A.5 and A.6, depending on the special case, there may still exist fixed points (in which case we proceed as above), or there is a line that maps onto itself, and by choosing the origin on that line, the remaining translation part $\mathbf{t}(t)$ is linear in t, and therefore contributes a constant vector to the velocity, so the statement is also true in those cases.

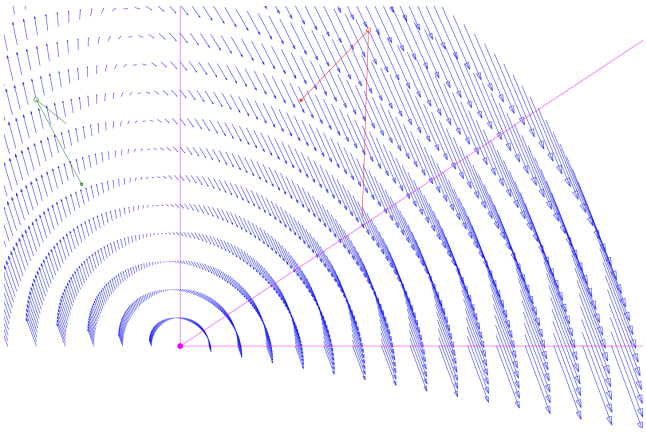

Fig. 8. A velocity field associated with an arbitrary SAM motion, that maps the green reference onto the red one. The resulting vector field is constant in time. Notice also how the velocity vectors vary linearly with the distance from the fixed point.

Figure 8 shows the vector field of the velocity at each point in the plane, associated with a SAM. The trajectories of points under this SAM are integral curves of this field. Since the field is stationary, they do not depend on time. In practice, given L, $\mathbf{q}$ and $\mathbf{t}$, the velocity vector at a point $\mathbf{p}$ may be computed as

$$\mathbf{p}' = \log(L)(\overrightarrow{\mathbf{qp}}) + \overrightarrow{\mathbf{t}_N} \tag{7}$$

where $\overrightarrow{\mathbf{t}_N}$ is the projection of $\overrightarrow{\mathbf{ot}}$ onto the eigenspace of L for eigenvalue 1.

6.3 Preservation of properties

Here we discuss quantities such that, if A preserves them, then also a SAM A_t preserves them for every time t.

6.3.1 *Preserving distance and angles (for isometries).* A is an isometry if L is an orthogonal matrix (here, with positive determinant, therefore equal to one). In this case SAM is either the identity, a translation, or it is a rotation (with a possible translation along its axis in the three dimensional case). The corresponding SAM will then be a constant-speed rotation (plus possibly a constant speed translation), and therefore an isometry for every t.

6.3.2 *Preserving angles (for similarities).* A similarity is a composition of a translation, a rotation, and a uniform scaling. The corresponding SAM will result from the composition of a constant speed rotation and translation and of an exponential uniform scaling, and is therefore also a similarity at every time value t.

6.3.3 *Preserving area (in 2D) and volume (in 3D).* For an affinity A to preserve volume (area in the case of an affinity in the plane), the necessary and sufficient condition is that $det(L) = 1$. Since $\det(e^B) = e^{\text{trace}B}$, we conclude that for a volume-preserving affinity A, $\text{trace}(\log(L)) = 0$, so $\text{trace}(t \log(L)) = t \cdot \text{trace}(\log(L)) = 0$, and hence $\det(L_t) = \det(e^{t \cdot \log(L)}) = e^{\text{trace}(t \cdot \log(L))} = 1$, which of course implies that L^t and hence A^t preserve area in 2D and volume in 3D.

6.4 Monotonic variation of volume (area)

When A does not preserve volume in 3D (or area in 2D), then A^t evolves it monotonically. As in (6.3.3) above, we notice

that $\text{trace}(\log(L)) = \log(\det(L))$ will be positive or negative depending on whether A increases or decreases volume. Then $\frac{d}{dt}\det(L(t)) = \text{trace}(\log(L))e^{t\cdot\text{trace}(\log(L))}$ has constant sign, and therefore the variation of volume is monotonic. Figure 9 shows a

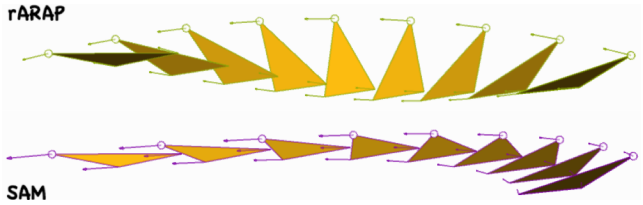

Fig. 9. A triangle evolving under rARAP and SAM. The lightness of color is proportional to area to show the monotonic evolution under SAM and the non-monotonic evoution under rARAP

comparison of rARAP and SAM in this respect. Notice how the area of the triangle grows and then shrinks back under rARAP (and hence also under ARAP), while it decreases monotonically for SAM.

7. SAM BENEFITS IN APPLICATIONS

In this section, we provide several examples of possible applications of SAM to the design of patterns and motions.

7.1 Regular patterns

When a continuous SAM is sampled at k time values separated by a constant Δ_t time interval, it produces *regular patterns* $F_k = A^{k\Delta_t}(F_0)$. Observing that $A^{(k+1)\Delta_t} = A^{\Delta_t} \circ A^{k\Delta_t}$, we see that in a regular pattern, each shape can be obtained by transforming the previous shape by a constant affinity ($M = A^{\Delta_t}$). Therefore, the steady pattern of shapes $F_i = M^i(F_0)$ is also defined by the initial shape F_0, the incremental affine transform M between each instance and the next one, and the repetition count n. Such constructions are often used in solid modeling, architectural design, and computer graphics (see for example [Jang and Rossignac 2009; van Emmerik et al. 1993; Wonka et al. 2003; Power et al. 1999; Larive and Gaildrat 2006; Alexander et al. 1977; Pauly et al. 2008]). Our EAR algorithm solves the reverse engineering problem of computing M from a given repetition count n and from the cumulative affinity $A = M^n$. Hence, we think of M as the n$^{\text{th}}$ root, $A^{\frac{1}{n}}$, of A. The ability to compute M as $A^{\frac{1}{n}}$ makes it possible to create regular patterns that interpolate the starting and ending poses F_0 and F_n of a shape. As such, it facilitates the precise design of these patterns, allowing the designer to manipulate the final instance $F_n = M^n(F_0)$ directly (rather than indirectly through manipulations of M or F_1) and also to change the repetition count n (see Figure 1). These two functionalities are essential for designing parametric models. Without this new ability of computing M from A and n, the designer would have to try guessing M by trial and error so that the last pose computed as $M^n(F_0)$ matches exactly the desired pose F_n. This process is nearly impossible to accomplish in practice, even in 2D, because the designer would have to simultaneously guess 6 variables (the coefficients of M). For example, assume that we let the designer manipulate the 3 vertices of a triangle F_1 that is the image $M(F_0)$ of an arbitrary triangle F_0 selected on the first shape. As the designer adjusts one of these three vertices, $M^n(F_0)$ changes in a non-linear and sometimes surprising fashion and there is absolutely no feedback to guide the designer as to the direction in which that vertex of $M(F_0)$ should go. At least not until the other two vertices are each exactly in their correct place... which of course creates a chicken and egg problem. This lack of separation makes it impossible for the designer to find the proper position of the three vertices of F_1. In contrast, if EAR is available, the designer may trivially drag the three vertices of $M^n(F_0)$ to match the vertices of the desired final pose F_n and EAR computes M automatically. To change the repetition count n without EAR, the designer would have to repeat the impossible guess work. With EAR, no further manipulation is necessary.

7.2 Sweeps

The envelope of the region swept by a moving surface S_t is the union of the grazing points [Abdel-Malek et al. 2006] of S_t, that is those points $\mathbf{p}$ of S_t whose instantaneous velocity is tangential to S_t. For general motions, the set of grazing points evolves on S_t with time. When the motion is approximated by a concatenation of screw motions, these grazing points remain constant during each screw span. This property was exploited in [Rossignac et al. 2007]. It can be extended to SAMs and polySAMs. For example, a characteristic point $\mathbf{q} + s\vec{\mathbf{t}}$ along a line through $\mathbf{q}$ with tangent $\vec{\mathbf{t}}$ on a plane or developable surface with surface normal $\vec{\mathbf{n}}$ satisfies $\vec{\mathbf{n}} \cdot (\log(L)(\overrightarrow{\mathbf{oq}} + s\vec{\mathbf{t}})) = \mathbf{o}$ yielding

$$s = \frac{\vec{\mathbf{n}} \cdot \log(L)(\overrightarrow{\mathbf{oq}})}{\vec{\mathbf{n}} \cdot \log(L)(\vec{\mathbf{t}})}. \quad (8)$$

To illustrate this property in 2D, consider an edge moving under the action of a SAM (see Figure 10). Using the previous formula,

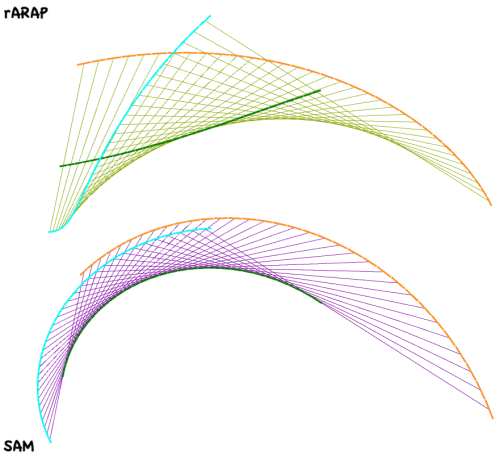

Fig. 10. SAM (bottom) has constant grazing points, so tracing them (green trajectory) yields directly portions of the envelope of the movement of a segment. In comparison, for rARAP (top), a point that is grazing at certain instant in time describes a trajectory (also in green) that is tangent to the envelope at that single time value.

we find (when it exists) the grazing point on the edge. Then, the boundary of the 2D region swept by the edge is a subset of the union of the initial and final poses and of the curves traced by the grazing point and by its two vertices.

7.3 Blending more than two poses

Our SAM construction may also be used to compute weighted averages of more than two poses or to average two steady motions.

As illustrated in Figure 11, the SAM interpolation may be used to produce two-dimensional (and even higher-dimensional) tensor-product patterns of 2D or 3D shapes.

To do so, we use SAM to produce a series of intermediate poses that interpolate between the top left (F_A) and the top right (F_B) control poses. We use SAM to generate the same number of intermediate poses in a series that interpolates between the bottom left (F_C) and the bottom right (F_D) control poses. Then, we consider pairs (top and bottom) of corresponding poses, compute a SAM between them and sample it using a series of intermediate poses (columns of shapes). We call this bi-directional pattern a *SAM patch*.

Even though all columns and the first and last rows are SAMs, the other rows are usually not steady. We can ensure that both the rows and the columns of the *SAM patch* are regular by computing one of the corner control poses from the other three. For instance, we may set $F_D = SAM(F_C, F_A, 1, F_B)$ (Figure 11).

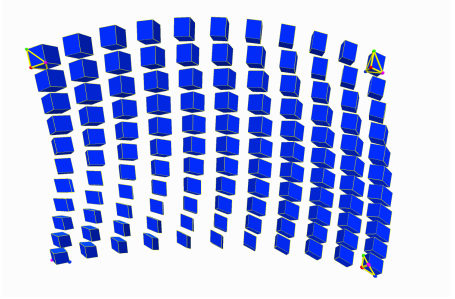

Fig. 11. A regular *SAM patch*.

SAM may be used as a new building block in a variety of subdivision or spline schemes that approximate or interpolate a series of control poses (see for example [Shoemake and Duff 1992] and [Hyun et al. 2002]).

For example, we can cascade SAM constructions to produce the equivalent for affinities of Bézier curves and four-point subdivisions. We have already discussed the quadratic Bézier motion (see Section 5.3). As a trivial extension, we show below, (Figure 12) a simple example of a cubic Bézier motion defined

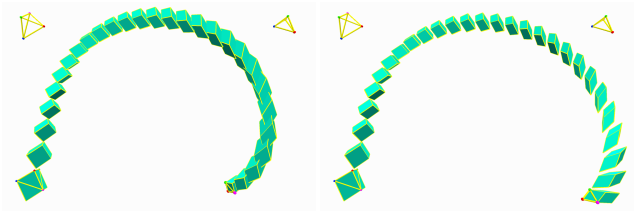

Fig. 12. Cubic Bézier SAM (left), and the result (right) of changing the bottom-right control pose.

by 4 control poses. The pose at time t is produced by the call `bezier(`F_A`,`F_B`,`F_C`,`F_D`,t)`, which is implemented, using the previously discussed functions, as:

```
Pose bezier(Pose F_A, Pose F_B,
            Pose F_C, Pose F_D, float t)
{ return SAM(bezier(F_A, F_B, F_C, t), t,
             bezier(F_B, F_C, F_D, t)); }
```

Although the resulting motion is not steady, it is smooth, pleasing, and easily controllable (as shown in Figure 12).

The ScrewBender scheme, developed by Powell and Rossignac [2008] to produce smooth rigid motions that approximate or interpolate a series of congruent poses, starts with a piecewise screw motion that interpolates pairs of consecutive congruent poses and subdivides that motion by modifying and inserting poses. All new poses are formulated by sampling screw motions that interpolate two original or previously computed poses. The same subdivision process may be used on a piecewise SAM interpolation of a series of affine (non-congruent) poses to subdivide it using the more general steady motions, rather than screw motions. Although the subdivision process will usually converge to a smooth and pleasing motion, the limit motion will in general not be steady. Nevertheless, the use of a SAM as the primitive interpolation for such constructions extends the domain beyond screw motions and may lead to steadier and hence more pleasing results than those produced when using unsteady motion primitives. The formal validation of such a subjective claim falls outside the scope of this paper.

Specifically, Figure 13 shows a PolySam interpolating four control poses (red blocks). The resulting motion is piecewise steady

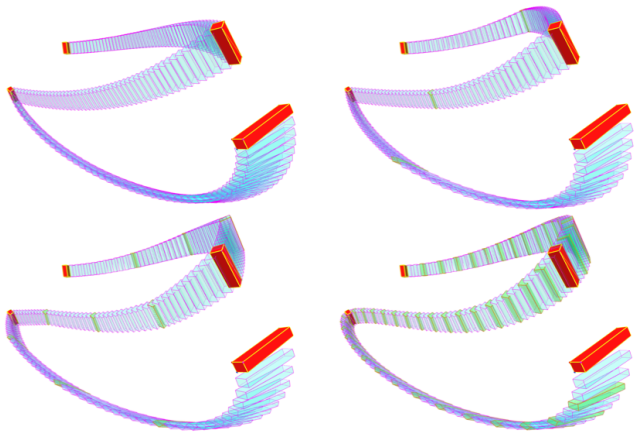

Fig. 13. The four control poses indicated by the red blocks define a continuous 3-span polySAM (top/left). We show it as a sequence of intermediate semi-transparent cyan poses. Notice the motion discontinuities at the control poses. Applying one four-point subdivision step inserts 3 new control poses (green blocks) that each splits a span in two (top/right). The new polySAM is different from the previous one, but still interpolates the original four control poses. Another subdivision step introduces 6 new control poses (bottom/let). The result of two additional subdivisions (bottom/right) produces a polySAM that appears perfectly smooth, both as a static pattern and when used to animate the motion of a shape. Yet, it is a concatenation of steady spans.

(three SAM spans), but has velocity discontinuities at the control poses. Successive applications of the four-point subdivision process in the space of affinities increases the number of SAM spans, but decreases the velocity discontinuities at span junctions. Our experiments suggest that 4 subdivision steps suffice to produce polySAMs that appear smooth.

Various formulae may be used to compute the intermediate vertices introduced by a four-point subdivision step. These formulae produce the same result (ignoring possible round-off errors). However, because here we are using SAM to average poses (and not

linear combinations to average vertices), these SAM averages do not commute and hence the choice of a particular formula affects the result. We have experimented with several constructions and report the one that we found to produce the most satisfactory results. Our construction of each new pose of the four-point subdivision essentially uses Neville's algorithm to evaluate the mid point of a cubic interpolant by averaging points on two quadratic interpolants.

Our implementation of the four-point subdivision is only a few lines of code (see below) that use as input n entries in the array `P` of poses. For clarity, we compute the subdivided sequence of $2n-1$ poses in a separate array `Q` and then copy them back into `P`.

```
void subdivide() {
  for(int i=1; i<n-1; i++) {
    F_L[i]=left(P[i-1],P[i],P[i+1]);
    F_R[i]=left(P[i+1],P[i],P[i-1]);
  }
  for(int i=0; i<n; i++) Q[2*i]=P[i];
  for(int i=1; i<n-2; i++)
      Q[2*i+1]=SAM(F_R[i],0.5,F_L[i+1]);
  Q[1]=F_L[1]; Q[2*n-3]=F_R[n-2];
  n=2*n-1;
  for(int i=0; i<n; i++) P[i]=Q[i];
}
```

Our implementation exploits the fact that each new control pose in the four-point scheme is the average of two parabolic SAM predictions, which we pre-compute and store in the temporary arrays of poses F_L and F_R. For each control pose, except the last, we compute such a left prediction (call to "`left`"). For each control pose, except the first, we compute the right prediction (using a call to "`left`", but passing the poses in reverse order). Then, for each span, we take the mid-course pose along the SAM (SAM average) between the right prediction of the left pose and of the left prediction of the right pose. We do not need to compute an average for the first and last spans, since each has only one prediction. For the parabolic prediction, function "`left`" returns a pose F_L at one quarter along a parabolic (quadratic Bézier) motion through 3 control poses (F_A, F_B, F_C), which we chose to construct as F_L`=left(`F_A`, `F_B`, `F_C`)` using

```
Pose left(Pose F_A, Pose F_B, Pose F_C) {
  return SAM(SAM(F_A, 0.625, F_B), F_C, .125, F_B);
}
```

This construction produces an interpolated pose, and is hence different from `leftControl`, which we used in Section 5.3 to construct a control pose.

Note that, in the construction above, we are using SAM in two places: (1) we are using SAM to predict the new poses at each subdivision step and (2) we are using SAM to interpolate the final set of poses by a piecewise steady motion (PolySAM). As discussed above, the latter use has well formulated advantages, over non steady alternatives, for subsequent motion processing. The benefits of the former use of SAM (as the fundamental interpolation primitive) include numeric robustness, speed, and the fact that if the original control poses are already forming a regular pattern, the subdivision process preserves steadiness. Although this simple polySAM four-point subdivision scheme consistently produces pleasant results, we have not performed a user study to compare their aesthetic quality to the quality of results produced by using unsteady interpolants or other subdivision schemes, which could be used (instead of our approach) to produce a dense set of control poses from which a polySAM could be constructed.

8. CONCLUSIONS

A motion A_t that interpolates between the identity $A_0 = \mathbb{1}$ and an affinity $A_1 = A$, is often needed to generate a continuous animation or a discrete set of evenly sampled poses that define a regular pattern of shapes. A variety of formulations have been proposed. The best choice may be dictated by subjective aesthetic criteria and by domain-dependent needs. In the absence of such criteria, when A is an isometry (rigid body transformation), a screw interpolation is often used because it produces a natural and pleasing motion that has a set of useful geometric properties. In two dimensions, the screw motion reduces to a pure rotation or a pure translation. When A is no longer restricted to be rigid, but is a planar similarity, the aesthetic and computational benefits of screw motions may be extended by using a logarithmic spiral interpolation, which includes the golden spiral famous for its beauty and which was named "spira mirabilis" (the miraculous spiral) because of its surprising and beautiful properties.

The Steady Affine Morph (SAM) introduced here unifies these solutions and elegantly extends the screw solution to arbitrary affinities in three dimensions and the two-dimensional logarithmic spiral solution to situations where A is not restricted to a similarity. A motion is steady when it satisfies $A_t = A^t$.

SAMs have several interesting properties. For example, (1) the affine relation between pairs of consecutive evenly spaced poses of a SAM is constant and (2) the instantaneous velocity of a point remains constant over time both in the global and the local (moving) frame. These properties result in pleasing curved trajectories of points and lower algebraic complexity when computing derived entities, such as the envelope of a swept volume.

The SAM solution proposed here differs from previously proposed approaches. For example, both the "as-rigid-as-possible" shape interpolation [Alexa et al. 2000] and the linear combination of logarithms proposed by Alexa [2002] are not steady.

We provide the implementation details and theoretical justification of our Extraction of Affinity Roots (EAR) algorithm, which uses closed form expressions to compute A^t in two and three dimensions, when it exists, and hence eliminates the performance, accuracy and stability shortcomings of previously proposed dimension-independent numeric solutions.

We show that solvability (i.e., the existence) of SAM is only affected by the amount of rotation and shear in a two-dimensional projection of the linear part L of the affinity. Furthermore, we show that SAM always exists when the rotation angle is less than $\pi/2$ and that the allowable angle for solvability contains a safe interval that grows to π as the shear factor grows from 0 to 1. We show that the SAM solution produced by EAR is the only solution satisfying $A_t = A^t$. We demonstrate the robustness of our solution using a direct manipulation environment, where the designer may edit the starting and ending poses. When no SAM exists, we propose unsteady or piecewise steady alternatives.

We believe that the simplicity and beauty of the Steady Affine Morph make it a prime candidate for designing motions and patterns and for representing local (instantaneous) approximations of more general (affine) motions, just as screw motions are used to locally approximate rigid motions in three dimensions.

REFERENCES

ABDEL-MALEK, K., BLACKMORE, D., AND JOY, K. 2006. Swept volumes: Foundations, perspectives, and applications. *International Journal of Shape Modeling 12,* 1 (June), 87–127.

AGOSTON, M. K. 2005. *Computer Graphics and Geometric Modeling: Mathematics*. Springer-Verlag.

ALEXA, M. 2002. Linear combination of transformations. In *SIGGRAPH 2002 Conference Proceedings*, J. Hughes, Ed. Annual Conference Series. ACM Press/ACM SIGGRAPH, 380–387.

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 157–164.

ALEXANDER, C., ISHIKAWA, S., AND SILVERSTEIN, M. 1977. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, New York.

ARNOLD, V. I. 1981. *Ordinary Differential Equations*. The MIT Press.

BARR, A. H., CURRIN, B., GABRIEL, S., AND HUGHES, J. F. 1992. Smooth interpolation of orientations with angular velocity constraints using quaternions. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. ACM Press, 313–320.

BREGLER, C., LOEB, L., CHUANG, E., AND DESHPANDE, H. 2002. Turning to the masters: motion capturing cartoons. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM, New York, NY, USA, 399–407.

BUSS AND FILLMORE. 2001a. Spherical averages and applications to spherical splines and interpolation. *ACMTG: ACM Transactions on Graphics 20*.

BUSS, S. R. AND FILLMORE, J. P. 2001b. Spherical averages and applications to spherical splines and interpolation. *ACM Trans. Graph. 20,* 2, 95–126.

CHENG, S. H., HIGHAM, N. J., KENNEY, C. S., AND LAUB, A. J. 2000. Approximating the logarithm of a matrix to specified accuracy. *SIAM J. Matrix Anal. Appl. 22,* 4, 1112–1125.

CHOI, J. AND SZYMCZAK, A. 2003. On coherent rotation angles for as-rigid-as-possible shape interpolation. Tech. rep., Georgia Institute of Technology.

CLIFFORD, W. 1882. *Mathematical Papers*. Macmillan, London.

CULVER, W. C. 1966. On the existence and uniqueness of the real logarithm of a matrix. *Proceedings of the American Mathematical Society 17,* 5 (October), 1146–1151.

DAVIES, P. I. AND HIGHAM, N. J. 2003. A schur-parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.*

DUFF, T. 1986. Splines in animation and modeling. In *SIGGRAPH '86 Course Notes on State of the Art Image Synthesis*. ACM Press.

EATON, J. W. J. W. 2000. *GNU Octave: a high-level interactive language for numerical computations: edition 3 for Octave version 2.0.13*. Network Theory Ltd., pub-NETWORK-THEORY:adr.

FU, H., TAI, C. L., AND AU, O. K. 2005. Morphing with laplacian coordinates and spatial-temporal texture. In *Pacific Graphics'05*. 100–102.

GOLUB, G. H. AND VAN LOAN, C. F. 1996. *Matrix Computations*, third ed. Johns Hopkins University Press, Baltimore, MD.

GRASSIA, F. S. 1998. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools: JGT 3,* 3, 29–48.

HIGHAM, D. J. AND HIGHAM, N. J. 2005. *MATLAB Guide*. SIAM.

HIGHAM, N. J. 1986. Newton's method for the matrix square root. *Math. of Comp.*.

HYUN, D.-E., JÜTTLER, B., AND KIM, M.-S. 2002. Minimizing the distortion of affine spline motions. *Graphical Models*.

JANG, J. AND ROSSIGNAC, J. 2009. Octor: Subset selection in recursive pattern hierarchies. *Graphical Models In Press, Corrected Proof*, –.

JOHNSTON, O. AND THOMAS, F. 1995. *The Illusion of Life: Disney Animation*. Disney Press.

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

KAVAN, L., COLLINS, S., ŽÁRA, J., AND O'SULLIVAN, C. 2008. Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph. 27,* 4, 105:1–105:23.

KAVAN, L. AND ŽÁRA, J. 2005. Spherical blend skinning: a real-time deformation of articulated models. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*. ACM, New York, NY, USA, 9–16.

KIM, B. AND ROSSIGNAC, J. 2003. Collision prediction. *J. Comput. Inf. Sci. Eng 3,* 4, 295–301.

KIM, M.-J., KIM, M.-S., AND SHIN, S. Y. 1995. A general construction scheme for unit quaternion curves with simple high order derivatives. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM Press, 369–376.

KIM, M. S. AND NAM, K. W. 1995. Interpolating solid orientations with circular blending quaternion curves. *Computer Aided Design 27*, 385–398.

LARIVE, M. AND GAILDRAT, V. 2006. Wall grammar for building generation. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*. ACM, New York, NY, USA, 429–437.

LEE, J. AND SHIN, S. Y. 2002. General construction of time-domain filters for orientation data. In *IEEE Transcations on Visualization and Computer Graphics*. Vol. 8(2). IEEE Computer Society, 119–128.

LLAMAS, I., KIM, B., GARGUS, J., ROSSIGNAC, J., AND SHAW, C. D. 2003. Twister: a space-warp operator for the two-handed editing of 3D shapes. In *Proceedings of ACM SIGGRAPH 2003*, J. Hodgins and J. C. Hart, Eds. ACM Transactions on Graphics, vol. 22(3). 663–668.

MA, L., CHAN, T. K. Y., AND JIANG, Z. 2000. Interpolating and approximating moving frames using B-splines. In *Pacific Conference on Computer Graphics and Applications*. IEEE Computer Society, 154–164.

MEINI, B. 2005. The matrix square root from a new functional perspective: Theoretical results and computational issues. *SIAM J. Matrix Anal. Appl. 26,* 2, 362–376.

MORTENSON, M. 2007. *Geometric Transformations for 3D Modeling*. Industrial Press Inc, New York.

PAULY, M., MITRA, N. J., WALLNER, J., POTTMANN, H., AND GUIBAS, L. 2008. Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics 27,* 3, #43, 1–11.

POWELL, A. AND ROSSIGNAC, J. 2008. ScrewBender: Smoothing piecewise helical motions. *IEEE Computer Graphics and Applications 28,* 1 (Jan./Feb.), 56–63.

POWER, J. L., BRUSH, A. J. B., PRUSINKIEWICZ, P., AND SALESIN, D. H. 1999. Interactive arrangement of botanical l-system models. In *I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*. ACM, New York, NY, USA, 175–182.

ROBERTS, K. S., BISHOP, G., AND GANAPATHY, S. K. 1988. Smooth interpolation of rotational matrices. In *Proceedings CVPR '88: Computer Vision and Pattern Recognition*. IEEE Computer Science Press, 724–729.

ROSSIGNAC, J. AND KIM, J. J. 2001. Computing and visualizing pose-interpolating 3D motions. *Computer-Aided Design 33,* 4, 279–291.

ROSSIGNAC, J., KIM, J. J., SONG, S. C., SUH, K. C., AND JOUNG, C. B. 2007. Boundary of the volume swept by a free-form solid in screw motion. *Computer-Aided Design 39,* 9, 745–755.

ROSSIGNAC, J. AND SCHAEFER, S. 2008. J-splines. *Computer-Aided Design 40,* 10-11, 1024–1032.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. ACM Press, 245–254.

SHOEMAKE, K. 1987. Quaternion calculus and fast animation. In *SIGGRAPH '87 Course Notes on State of the Art Image Synthesis*. ACM Press, 101–121.

SHOEMAKE, K. AND DUFF, T. 1992. Matrix animation and polar decomposition. In *Proceedings of Graphics interface '92*. Morgan Kaufmann Publishers Inc., 258–264.

SORKINE, O. AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. 109–116.

VAN EMMERIK, M., RAPPOPORT, A., AND ROSSIGNAC, J. 1993. Simplifying interactive design of solid models: A hypertext approach. *The Visual Computer 9,* 5 (Mar.), 239–254.

WANG, W. AND JOE, B. 1993. Orientation interpolation in quaternion space using spherical biarcs. In *Graphics Interface '93*. 24–32.

WHITED, B., NORIS, G., SIMMONS, M., SUMNER, R. W., GROSS, M., AND ROSSIGNAC, J. 2010. Betweenit: An interactive tool for tight inbetweening. *Comput. Graph. Forum 29,* 2 (May), 605–614.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*. ACM, New York, NY, USA, 669–677.

ZHANG, S., NEALEN, A., AND METAXAS, D. 2010. Skeleton Based As-Rigid-As-Possible Volume Modeling. H. P. A. Lensch and S. Seipel, Eds. Eurographics Association, Norrköping, Sweden, 21–24.

APPENDIX

This Appendix presents our Extraction of Affinity Roots (EAR) algorithm, which computes A^t, when it exists as a real matrix. We include the derivation of the closed form expression used by EAR. We start by proving the formulation of A^t in terms of $\log(L)$. Then, we present a closed form expression for the exponential of a 2×2 matrix C and explain how we exploit it for deriving the closed form expression for $\log(L)$ in 2D. We also present the simple algorithm for the case-selection process for the EAR algorithm in 2D. Finally, we discuss how the 2D solution is used for solving the 3D problem. Both expositions of the 2D and 3D solutions incorporate all special cases.

A.1 Reduction to a linear problem

In our solution, we convert the problem of computing A^t to one of computing L^t . Recall that $A(\mathbf{p}) = L(\overrightarrow{\mathbf{op}}) + \mathbf{t}$. In a general configuration, A has a unique fixed point $\mathbf{q}$. Singular cases, when it does not, require custom treatment as discussed in Section A.5. The fixed point then satisfies $\mathbf{q} = L(\overrightarrow{\mathbf{oq}}) + \mathbf{t}$. Eliminating $\mathbf{t}$ from these two equations and exploiting the linearity of L, one obtains $A(\mathbf{p}) = L(\overrightarrow{\mathbf{qp}}) + \mathbf{q}$. Thus, for simplicity of exposition, we will usually refer to a coordinate system with origin at $\mathbf{q}$.

Hence, given the fixed point $\mathbf{q}$, we express the effect of SAM as

$$A^t(\mathbf{p}) = L^t(\overrightarrow{\mathbf{qp}}) + \mathbf{q} = e^{t\log(L)}(\overrightarrow{\mathbf{qp}}) + \mathbf{q} \tag{9}$$

Our EAR algorithm provides a closed-form expression for A^t and for the fixed point $\mathbf{q}$ when they exist. Note that, when $\log(L)$ is real, for each time t, we can —if desired— recover the 4×3 matrix form of A^t as $A^t = [L^t \mid \mathbf{t}_t] = [e^{t\log(L)} \mid \mathbf{q} - L^t(\overrightarrow{\mathbf{oq}})]$.

A.2 Closed form for e^{tC} in 2D

Let

$$C = \begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix}. \tag{10}$$

Let $d_C = \left(\frac{c_1+c_4}{2}\right)^2 - (c_1c_4 - c_2c_3)$ and $z = (c_1 - c_4)/2$. d_C is the discriminant of the characteristic polynomial of C, so its sign determines the nature (real or complex) of the eigenvalues, and the test d_C==0 determines their algebraic multiplicity. In what follows, we assume that $d_C \neq 0$, for simplicity. The special circumstance where $d_C = 0$ (and therefore where there is a double real eigenvalue) can be considered separately, as we do in the algorithm presented later in Sub-section A.5.

Hence, the —possibly complex— eigenvalues of C are

$$\lambda_{1,2} = \frac{c_1 + c_4}{2} \pm \sqrt{d_C}.$$

We form a matrix having as columns the linearly independent eigenvectors of C:

$$V = \begin{bmatrix} \frac{c_2}{\sqrt{d_C}-z} & \frac{-c_2}{\sqrt{d_C}+z} \\ 1 & 1 \end{bmatrix}.$$

Then $\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = V^{-1} \cdot C \cdot V$, and therefore, since the exponentiation commutes with a change of basis (because of telescopic cancellation in the series expansion), we can compute the exponential of C as:

$$e^{tC} = V \cdot \begin{bmatrix} e^{t\lambda_1} & 0 \\ 0 & e^{t\lambda_2} \end{bmatrix} \cdot V^{-1} =$$

$$\frac{1}{2}\begin{bmatrix} \frac{(\sqrt{d_C}+z)e^{\lambda_1 t}+(\sqrt{d_C}-z)e^{\lambda_2 t}}{\sqrt{d_C}} & \frac{c_2}{\sqrt{d_C}}\left(e^{\lambda_1 t} - e^{\lambda_2 t}\right) \\ \frac{c_3}{\sqrt{d_C}}\left(e^{\lambda_1 t} - e^{\lambda_2 t}\right) & \frac{(\sqrt{d_C}-z)e^{\lambda_1 t}+(\sqrt{d_C}+z)e^{\lambda_2 t}}{\sqrt{d_C}} \end{bmatrix}. \tag{11}$$

This is the expression used in our code for computing the exponential of a matrix C. Notice that for real c_i —although derived using possibly complex eigenvalues— this expression always yields a real result: when $d_C < 0$, λ_1 and λ_2 are complex conjugates, so the sum of their exponentials is real, and the difference is purely imaginary. For the implementation, we distinguish these two situations, and handle them separately to keep the computation completely within the real domain.

While we used general complex eigenvectors to obtain this formula, one can check that in the special cases (when c_2 and/or c_3 are zero) the resulting formula is still correct.

A.3 Computing $\log(L)$ in 2D

Using equation (11) in the previous section, we set out now to find a closed formula for $\log(L)$ of a 2×2 real matrix L:

$$L = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix}. \tag{12}$$

With the same notation as before, we want now to compute the four entries c_i of a matrix C such that

$$C = \begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix} = \log(L),$$

which is to say such that $e^C = L$, which using equation (11) with $t = 1$ yields four non-linear equations for the c_i.

To simplify the derivation of a closed form expression for the solution of this system of equations, we introduce new variables: $x = (c_1 + c_4)/2$, $y = \sqrt{d_C} = \sqrt{x^2 - det(C)}$, and $r = (a_{11} + a_{22})/2 = \text{trace}(L)/2$.

Letting $t = 1$ in Equation (11), and equating it to $L = [a_{ij}]$, we obtain

$$z = \frac{(a_{11} - a_{22})c_2}{2a_{12}}, \tag{13}$$

and

$$c_3 = \frac{c_2 a_{21}}{a_{12}}. \tag{14}$$

Furthermore, summing the equations for a_{11} and for a_{22} and simplifying, we find that

$$e^x = \frac{a_{11} + a_{22}}{e^y + e^{-y}}. \tag{15}$$

Next, we notice that $y^2 = z^2 + c_2 c_3$, and substituting z and c_3 using (13) and (14), we obtain

$$c_2 = \frac{2a_{12}y}{\sqrt{(a_{11} - a_{22})^2 + 4a_{21}a_{12}}}. \tag{16}$$

Finally, using (15) and (16) to substitute in the equation for a_{12}, we obtain an equation that expresses y in terms of the coefficients of L:

$$1 = \frac{(a_{11} + a_{22})(e^y - e^{-y})}{\sqrt{(a_{11} - a_{22})^2 + 4a_{21}a_{12}}(e^y + e^{-y})}. \tag{17}$$

To solve this equation, we need to distinguish two situations: (1) when the expression under the square root is strictly negative (treated by function `G1` in Algorithm 1 below), and (2) when it is positive. The latter situation is further distinguished into cases `G2` and `G3`. We discuss all three below.

`G1` (L **has two complex conjugate eigenvalues**): Here we set $y = \theta i$ and obtain

$$\tan(\theta) = \frac{\sqrt{\det(L) - r^2}}{r}. \tag{18}$$

Substituting backwards $y = i\theta$ in equations (15), respectively (16), (14), and (13), we find values for x, respectively c_2, c_3, and z, and then obtain c_1 and c_4, using

$$c_1 = x + z \quad \text{and} \quad c_4 = x - z. \tag{19}$$

`G2` (L **has real eigenvalues**): Here, the expression under the square root in (17) is positive, and, after writing the formula in terms of e^{2y} only, we obtain:

$$y = \frac{\log\left(\frac{\chi+1}{\chi-1}\right)}{2}, \quad \text{where} \quad \chi = \frac{r}{\sqrt{r^2 - det(L)}}, \tag{20}$$

and again substitute in the previous equations (16), (14), (15) and (13) to obtain all the c_i. We distinguish a special singular case `S2` when both eigenvalues of L are negative and identical, and there exist two linearly independent eigenvectors. Then L is a rotation by 180 degrees (plus perhaps a uniform scaling) and the logarithm is $\begin{bmatrix} \log(-s) & -\pi \\ \pi & \log(-s) \end{bmatrix}$, where s is the scaling factor.

`G3` (**no SAM**): In configurations with negative eigenvalues, the logarithm does not exist (see Section 5.1), nor does it make sense to speak of A^t for any non-integer t. In this case, there is no SAM. Note that the situation where the eigenvalues are real and have different signs is not considered, since we are dealing only with orientation-preserving affinities.

With these closed-form expressions at our avail, given a real matrix L, we can now compute L^t by finding C and then using equation (11) to compute $L^t = e^{t\log(L)} = e^{tC}$.

A.4 Singular cases in 2D

We observe that, when 1 is not an eigenvalue of L, equation $\mathbf{q} = L(\overrightarrow{\mathbf{oq}}) + \mathbf{t}$, which we rewrite as $\big(L - \mathbb{1}\big)(\overrightarrow{\mathbf{oq}}) = \overrightarrow{\mathbf{ot}}$, has a unique solution, and therefore the affinity A has a unique fixed point $\mathbf{q}$. Expressing A in a basis that has that fixed point as origin nullifies the translation part. This situation accounts for the vast majority of affinities. Exceptional situations where 1 is an eigenvalue of L, gives rise to three singular cases:

(1) $L = \mathbb{1}$: Here, the affinity A is a translation by $\overrightarrow{\mathbf{ot}}$ (no fixed points). In this case SAM is given by $A^t = \big[\mathbb{1}\,\big|\,\mathbf{o} + t\overrightarrow{\mathbf{ot}}\big]$.

(2) 1 is a simple eigenvalue of L: Here, there are two linearly independent vectors $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$ with eigenvalues 1 and $\lambda \neq 1$ respectively, and L is a scaling in only one direction. In this case, L^t is a scaling by λ^t in the direction of $\overrightarrow{e_2}$, as is seen easily by diagonalizing L. The translation part of the affinity is $\overrightarrow{\mathbf{ot}} = \tau_1\overrightarrow{e_1} + \tau_2\overrightarrow{e_2}$. The affinity $A' = [L\,|\,\mathbf{o} + \tau_2\overrightarrow{e_2}]$ has a full "fixed" line, parallel to $\overrightarrow{e_1}$, such that points on the line map onto other points on the line (by a constant translation). Choosing the origin at any point on that line will give a representation of A where $\tau_2 = 0$. Notice that A is the composition of A' and a pure translation. Hence, in this case, SAM is an exponential scaling in the direction of $\overrightarrow{e_2}$ and a constant-speed translation in the direction of $\overrightarrow{e_1}$, so that the displacement at time t is $t\tau_1\overrightarrow{e_1}$.

(3) 1 is a double eigenvalue, and L is a shear (there is only one eigenvector of eigenvalue 1): This case is similar to the previous one; there is an eigenvector $\overrightarrow{e_1}$ of eigenvalue 1, and a linearly-independent generalized eigenvector $\overrightarrow{e_2}$ such that $\big(L - \mathbb{1}\big)(\overrightarrow{e_2}) = \overrightarrow{e_1}$. Constructing A' as in the previous case, we again find a fixed line under the action of A, parallel to $\overrightarrow{e_1}$. The same decomposition as discussed above yields an exponential that takes care of A' and an additional translation in the direction of $\overrightarrow{e_1}$.

A.5 2D case selection

Given a 2×2 matrix L, we determine the corresponding singular or general case as follows.

We first compute

$$r = \frac{\text{trace}(L)}{2}$$

and the discriminant of the characteristic polynomial of L:

$$d_L = r^2 - \det(L).$$

The sign of d_L indicates if the eigenvalues of L are real or imaginary, and if real ($d_L \geq 0$), if they are double ($d_L = 0$) or not.

We also compute

$$s = \sqrt{d_L},$$

so that the eigenvalues of L are $r + s$ and $r - s$. We can now select the proper case using the decision logic described in Algorithm 1. In the algorithm, the procedures `S1` and `S2` correspond to the singular configurations in two dimensions.

`S1` is invoked when L has 1 as an eigenvalue ($d_L \geq 0$ and either $r + s$ or $r - s$ equal 1). It uses the rationale in the previous section to find a solution even though in this case there may be infinitely many or no fixed points.

`S2` id invoked when L is the composition of a rotation by π radians and a uniform scaling, or in other words, when it is a uniform

Algorithm 1 EAR algorithm in 2D

```
if (d_L < 0) G1();
else if (r > s){
  if (r + s != 1 && r - s != 1) G2();
  else S1();
}
else if ( s == 0 && a_12 == 0 && a_21 == 0) S2();
else G3();
```

scaling by a negative factor, in which case we directly apply the solution discussed in Section A.3.

The procedures `G1`, `G2`, and `G3` correspond to the general configurations in two dimensions.

`G1` is invoked when L has two complex conjugate eigenvalues (which is implied by $d_L < 0$). Notice that d_L is the quantity under the square root in equation (17), so we can perform a change of variables $y = \theta i$ as before, and compute θ, x, c_2, c_3, z and ultimately c_1 and c_2 following the steps discussed in Section A.3 for this case.

`G2` is invoked when L has two positive eigenvalues ($d_L \geq 0$ and $r \geq s$) and none of them is equal to 1 ($r + s \neq 1$ and $r - s \neq 1$). In this case, no change of variable is needed, and one may compute y directly from equation (17) as in equation (20), and then again solve backwards and obtain C as in Section A.3.

`G3` corresponds to the general case where there is no real logarithm of L, and hence there is no SAM. This happens when $d_L > 0$ and $r < s$, which corresponds to the complement of the union of all the other cases (r cannot be exactly equal to s because then one eigenvalue would be zero and the matrix singular). Strategies for computing the motion in this case are discussed in section 5.

A.6 SAM in 3D

As in the 2D case, to compute A^t in 3D, we distinguish general configurations and singular ones.

The explicit computation of A^t in two dimensions hinges on having simple closed-form expressions for the eigenvalues and eigenvectors of 2×2 matrices. We have attempted first to attack the 3D problem using the same strategy, but quickly realized that the 3D version is significantly more complicated. The eigenvalues are now the roots of a cubic, and hence using their algebraic expression to find a closed form solution for the exponential appears beyond reach.

Consequently, our solution of the 3D problem uses a different approach. The basic observation is that, in 3D, L always has at least one real eigenvalue with an associated eigenvector. Therefore, the problem may be split into a scaling along the direction of that eigenvector, and a two-dimensional SAM in the directions of the other (possibly generic) eigenvectors. In this way, the problem is clearly solved for all the general cases, and all that really needs to be carefully discussed are the singular cases where some eigenvalue is one, which present a more diverse set of sub-cases than in the two-dimensional problem.

As before, computing the fixed points of A amounts to solving the system of three linear equations $A(\mathbf{q}) = \mathbf{q}$ or equivalently the system

$$(L - \mathbb{1})(\overrightarrow{\mathbf{oq}}) = -\overrightarrow{\mathbf{ot}}. \qquad (21)$$

Therefore, the existence and uniqueness of solutions depends on whether 1 is an eigenvalue of L or not, and on the relationships between the eigenspaces of eigenvalue 1 and the non-homogeneous part of the system (given by the translation part $\mathbf{t}$ of the affinity). The most general situation is when no eigenvalue is exactly equal to 1. Choosing a reference where the origin lies at the unique fixed point, in this case, eliminates the translation part, and the strategy above reduces the problem to solving a two dimensional SAM using the approach presented above.

Otherwise, let $\overrightarrow{e_1}$ be an eigenvector of eigenvalue 1, and let $\vec{v} = \overrightarrow{\mathbf{ot}}$ be the translation vector of the affinity. Recall that fixed points need to satisfy Equation (21). In this case $\mathbb{R}^3$ decomposes in two subspaces: the kernel, or null-space, $N = Ker(L - \mathbb{1})$ (the space of vectors whose image by $L - \mathbb{1}$ is the zero vector) and the co-kernel N^C (which satisfies that $(L - \mathbb{1})(N^C) \subseteq N^C$), so that $\vec{v} = \vec{v}_N + \vec{v}_{N^C}$. If $\vec{v}_N = \overrightarrow{(0,0,0)}$, then the linear system (21) can still be solved (although the solution is not unique) and choosing the origin at any solution point $\mathbf{q}$ annihilates the translation part, so the solution of the general case may be applied. As in the two-dimensional case, when $\vec{v}_N \neq \overrightarrow{(0,0,0)}$, we may consider the affinity $A'(\mathbf{p}) = A(\mathbf{p}) - \vec{v}_N$, which can be solved as above, and add to it a translation with constant velocity $\vec{v}_N$.

Notice that, when we carry out computations in the basis given by the (possibly generalized) eigenvectors, and since the matrix need not be symmetric, the eigenvectors are not necessarily orthogonal to each other, and the matrix of eigenvectors may be ill conditioned. Nonetheless, our reference implementation is able to compute a solution even for very large condition numbers (in the order or 10^{15}). These extreme cases, however, where purposely constructed for testing, and are hard to produce when interacting with the application directly.