Rigid Shape Interpolation Using Normal Equations

William Baxter* OLM Digital, Inc.
Pascal Barla† INRIA Bordeaux University
Ken-ichi Anjyo‡ OLM Digital, Inc.

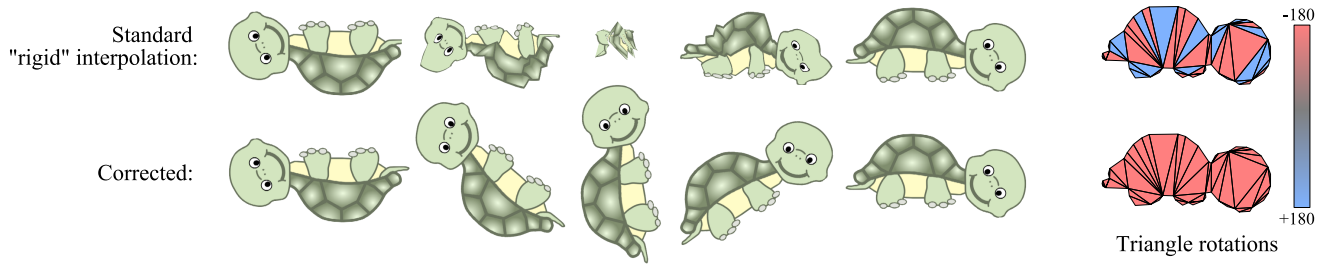

Figure 1: ***Rigid Morphing with large rotations:*** *For small enough rotations the "as-rigid-as-possible" technique generates high-quality interpolations. However, large rotations can lead to inconsistencies that cause artifacts such as those shown here (top). Our technique yields consistent rotations, resulting in the expected rigid interpolation (bottom).*

Abstract

In this paper we provide a new compact formulation of rigid shape interpolation in terms of normal equations, and propose several enhancements to previous techniques. Specifically, we propose 1) a way to improve *mesh independence*, making the interpolation result less influenced by variations in tessellation, 2) a faster way to make the interpolation *symmetric*, and 3) simple modifications to enable *controllable* interpolation. Finally we also identify 4) a failure mode related to *large rotations* that is easily triggered in practical use, and we present a solution for this as well.

Keywords: npr, interpolation, morphing, in-betweening, least-squares, Poisson problem

1 Introduction

Two-dimensional morphing and shape interpolation algorithms have many practical applications. Algorithms for generating 2D morphs are commonly found in commercial video editing software such as Adobe After Effects as well as packages for creating web-oriented or cartoon animations, such as Adobe Flash or Toon Boom. Animated two dimensional shapes also have applications for real-time simulation of large crowds [Kavan et al. 2008].

Recently, a variety of new algorithms that preserve rigidity [Alexa et al. 2000; Sheffer and Kraevoy 2004; Fu et al. 2005; Xu et al. 2005; Sumner et al. 2005] have been introduced. These rigidity-preserving methods generate interpolations which have a very natural, almost physical, appearance, and do so in a very efficient manner, making them suitable for interactive applications. Rigidity preserving interpolation works well and is a very practical way to generate high-quality shape interpolations. However there are a number of limitations in practice which we address here:

- **Mesh independence:** To the extent possible, the results should be independent of the tessellation of the underlying mesh used. This property is lacking in [Gotsman and Surazhsky 2001; Surazhsky and Gotsman 2001; Alexa et al. 2000].
- **Symmetry:** It is often desirable for an interpolation to be symmetric in its arguments. That is, the vertex paths generated interpolating from shape A to shape B should be the same as from B to A. One solution is presented in [Alexa et al. 2000], but it increases the computation load of the algorithm tremendously.
- **Controllability:** once the interpolation has been computed, it is desirable to be able to tweak it in some manner if it does not generate exactly the animation desired. Controllability was not addressed in the previous work on rigid interpolation ([Alexa et al. 2000; Xu et al. 2005]).
- **Rotational consistency:** For large rotations (>180 degrees), which are common in practical 2D animations, rotational inconsistencies can occur causing severe artifacts. This affects both [Alexa et al. 2000] and [Xu et al. 2005].

We provide a solution to each of these issues in this paper. Furthermore, in support of our explanation of these solutions, we provide an alternate presentation of the mathematics behind rigid interpolation in terms of normal equations. We believe the normal equations offer a more intuitive formulation, and also make the path to implementation straightforward. Our derivation follows that of [Alexa et al. 2000] most closely, since we find it to be the most accessible and since, similar to that work, we are also primarily interested in 2D. The technique of [Xu et al. 2005] is closely related, however, and we discuss this connection in Section 3.2.

2 Previous Work

The term "morphing" was introduced by [Beier and Neely 1992] to describe a combination of image-space warping with cross-dissolve between pixels. Many other image-space morphing algorithms were introduced subsequently with many refinements, a good example being that of [Lee et al. 1995]. A more complete survey of image-space moprhing can be found in [Wolberg 1998]. One problem with image-space morphing algorithms is that they have

*e-mail: baxter@olm.co.jp
†e-mail: pascal.barla@labri.fr
‡e-mail: anjyo@olm.co.jp

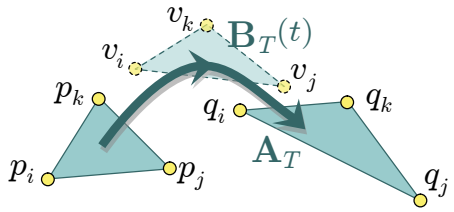

Figure 2: ***Triangle mapping notation:*** *The Jacobian, $\mathbf{A}_\mathcal{T}$, of the mapping between two triangles. We define $\mathbf{A}_\mathcal{T}(t)$ to be the* desired *interpolated Jacobian at parameter t, and $\mathbf{B}_\mathcal{T}(t)$ to be that actually obtained by the interpolation.*

difficulty handling large relative rotations in an image, resulting in unnaturally distorted geometry, self-intersections, and/or local orientation reversals ("flips"). It is also difficult to deal with images where foreground and background should behave differently.

Object-space morphing is an alternative which addresses some of those issues. A variety of techniques have been presented that operate on both 2D and 3D geometry, e.g. [Gregory et al. 1998; Lee et al. 1999; Kanai et al. 2000]. A good, though slightly dated, survey can be found in [Alexa 2002]. Some works have advanced the state of the art of object space-morphing by guaranteeing intersection-free solutions to the vertex path problem [Gotsman and Surazhsky 2001; Surazhsky and Gotsman 2001].

Recently a great deal of attention has focused on methods which attempt to preserve rigidity, or local differential quantities [Alexa et al. 2000; Sheffer and Kraevoy 2004; Xu et al. 2005; Fu et al. 2005; Sumner et al. 2005]. These mesh-based object-space algorithms solve a vertex optimization problem to find intermediate shapes that introduce as little local shear and stretch as possible. The work of [Schaefer et al. 2006] is unique in that it introduces an image-space rigidity-preserving deformation method, though it suffers from several of the same problems of other image space methods. However, in general, these newer rigidity-preserving, differential coordinate methods handle rotations very naturally, produce low distortion, and maintain a low runtime cost. Nevertheless there are some issues that remain with these methods, as mentioned in the introduction, and addressing these issues is the focus of our work.

Recently we became aware of work by Choi and Szymczak [2003] by way of [Fu et al. 2005] who also tackled the rotation consistency problem (called "rotation coherence" in their work) in a manner very similar to what we propose here. Their work also provides a detailed proof that the procedure we describe will succeed whenever a consistent rotation exists. We offer a small improvement over their method by taking an additional step to ensure that the rotation assignment chosen is unique and minimal. On the other hand their method is slightly more succinct than ours, made so by the elimination of the special boundary handling step.

3 Background

3.1 "As-rigid-as-possible" interpolation

Alexa et al. [2000] presents rigidity-preserving interpolation as a quadratic minimization problem. The problem is as follows: given two meshes $\mathcal{P}$ and $\mathcal{Q}$ composed of corresponding triangles $\mathcal{P}_\mathcal{T}$ and $\mathcal{Q}_\mathcal{T}$ for $\mathcal{T} = 1 \ldots M$, find an interpolation that preserves rigidity. The procedure proposed is the following. For each triangle pair, compute the Jacobian of the affine transformation that maps one triangle to the other, $\mathbf{A}_\mathcal{T}$ (Fig. 2), and interpolate these individual matrices independently to get $\mathbf{A}_\mathcal{T}(t)$, with $\mathbf{A}_\mathcal{T}(0) = \mathbf{I}$ and $\mathbf{A}_\mathcal{T}(1) = \mathbf{A}_\mathcal{T}$. This is done using a polar decomposition, and linearly interpolating rotation and shear components independently. Specifically, if the polar decomposition of $\mathbf{A}_\mathcal{T}$ is $\mathbf{R}(\alpha)\mathbf{S}$, then $\mathbf{A}_\mathcal{T}(t) = \mathbf{R}(t\alpha)((1-t)\mathbf{I} + t\mathbf{S})$, where α is a rotation angle extracted from the orthogonal part of the polar decomposition. Finally, the key idea is to find intermediate vertex trajectories $v_i(t)$ by minimizing a quadratic error function:

$$E(t) = \sum_{\mathcal{T}=1}^{M} \|\mathbf{B}_\mathcal{T}(t) - \mathbf{A}_\mathcal{T}(t)\|^2 \qquad (1)$$

where $\|\cdot\|$ is the Frobenius norm and $\mathbf{B}_\mathcal{T}(t)$ are the transformations that relate the initial vertices of $\mathcal{P}_\mathcal{T}$ to unknown vertices $v_i(t)$.

As explained in [Alexa et al. 2000], by taking the derivative of (1), setting it to zero, and rearranging all the terms, a linear system of the form $V(t) = -\mathbf{H}^{-1}\mathbf{G}(t)$ can be obtained where $V(t)$ is a vector containing all the unknowns, and $\mathbf{H}$ is a constant matrix independent of t. Though straightforward, the steps required to get from (1) to the final linear system using direct differentiation are laborious and yield little insight. Consequently, the intermediate steps are omitted in [Alexa et al. 2000]. However, by expressing the problem instead in terms of least squares normal equations, in Section 4 we are able to provide a complete and succinct end-to-end derivation that reveals more clearly the structure of the linear system.

One key aspect of the differential coordinate formulations like [Alexa et al. 2000; Xu et al. 2005; Sumner et al. 2005] is that they result in linear systems for which the left hand side (LHS, here $\mathbf{H}$) is independent of t. Thus the interpolation problem can be solved very efficiently by LU-decomposing the LHS once for the whole interpolation, and finding solutions for each given t by quick LU back-substitution. This advantage is lost, however, when symmetrizing the cost function as proposed in [Alexa et al. 2000]. In Section 5 we show that a symmetric cost can be achieved with little additional per-t cost.

3.2 Comparison with Poisson interpolation

Xu et al. [2005] suggest an alternate way to generate rigidity-preserving interpolations, by solving a Poisson problem on a domain mesh. This method is, in fact, nearly identical to that of [Alexa et al. 2000]. The gradients taken by Xu et al. are those of the coordinate functions: $\nabla\mathbf{x}(x,y), \nabla\mathbf{y}(x,y)$. By definition, these gradients stacked together form the Jacobian. Xu et al. acknowledge this, however, they claim that their solution differs from that of Alexa et al. in that, in contrast to the Frobenius norm, their formulation "directly minimizes least squares differences between [gradient] vectors using the L^2 norm". However, that is precisely what the Frobenius norm gives as well. There is one difference between the techniques, however: weighting. Relying on [Tong et al. 2003], Xu et al. note that their discrete Poisson formulation is equivalent to finding S that minimizes $\sum_\mathcal{T} a_\mathcal{T}\|\nabla S - G\|^2$ (Xu et al. Eq. 10), where $a_\mathcal{T}$ is the area of triangle $\mathcal{P}_\mathcal{T}$. In our notation, $\mathbf{B}_\mathcal{T}(t) \equiv \nabla S$ and $\mathbf{A}_\mathcal{T}(t) \equiv G$, so this is precisely (1) with an additional weighting factor of $a_\mathcal{T}$. If we introduce this weighting in (1) then, for 2D problems, as-rigid-as-possible interpolation becomes identical to Poisson shape interpolation. Without the weighting, the results of Alexa et al. can be seen to depend significantly on the tessellation (see Figure 3).

Xu et al. made a significant contribution in the form of providing a more rigorous and general mathematical foundation for rigid interpolation by relating it to the Poisson problem. However, given that solving the Poisson problem involves second order differential

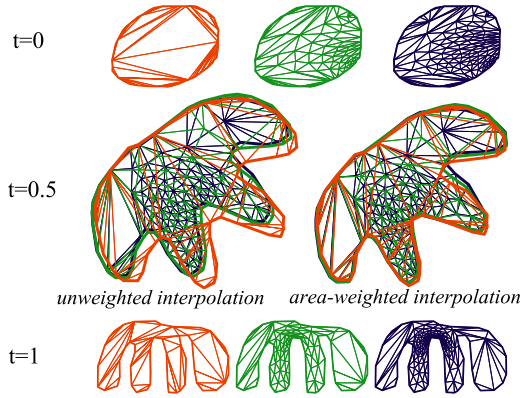

Figure 3: ***Tessellation dependence****: Formulating interpolation as unweighted least squares as in (1) leads to tessellation dependence (middle-left). Area-weighted least squares (3) makes the behavior more uniform with respect to tessellation (middle-right).*

quantities vs. only first order for least squares, and given the relative complexities of the two approaches, we believe that it is more logical to formulate rigid interpolation in the manner of Alexa et al., as a least squares problem. Furthermore, we feel the interpretation of Alexa et al. is easier to follow and carries less extraneous mathematical baggage that is not required to understand or implement the method.

Finally we note that, because the same polar decomposition is used for extracting and interpolating rotations in both methods, Poisson shape interpolation (both 2D and 3D) also suffers from the same rotational inconsistency problems we discuss in Section 7.

4 Area-weighted ARAP interpolation

Next we present a re-formulation of the math of [Alexa et al. 2000], expressed in terms of normal equations rather than direct minimization of quadratic error. The result is the same—mathematically the two approaches are equivalent. However, the least squares formulation is more succinct, allowing us to enumerate all the steps necessary to arrive at the final linear system. An overview of least squares and normal equations can be found in Appendix A.

Let $\mathbf{A}_\mathcal{T}$ be the Jacobian of the affine map that relates triangle $\mathcal{P}_\mathcal{T} = \{p_i, p_j, p_k\}$ to triangle $\mathcal{Q}_\mathcal{T} = \{q_i, q_j, q_k\}$ (Fig. 2). In terms of x and y coordinates, $\mathbf{A}_\mathcal{T}$ can be computed as:

$$\mathbf{A}_\mathcal{T} = \begin{bmatrix} p_i^x - p_k^x & p_i^y - p_k^y \\ p_j^x - p_k^x & p_j^y - p_k^y \end{bmatrix}^{-1} \begin{bmatrix} q_i^x - q_k^x & q_i^y - q_k^y \\ q_j^x - q_k^x & q_j^y - q_k^y \end{bmatrix}. \quad (2)$$

Making the following additional definitions

$$\mathbf{P}_\mathcal{T} = \begin{bmatrix} p_i^x & p_i^y \\ p_j^x & p_j^y \\ p_k^x & p_k^y \end{bmatrix}; \ \mathbf{Q}_\mathcal{T} = \begin{bmatrix} q_i^x & q_i^y \\ q_j^x & q_j^y \\ q_k^x & q_k^y \end{bmatrix}; \ \mathbf{D} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$

$$\mathbf{P}_\mathcal{T}^* = (\mathbf{D}\mathbf{P}_\mathcal{T})^{-1}\mathbf{D}$$

we can rewrite (2) as $\mathbf{A}_\mathcal{T} = \mathbf{P}_\mathcal{T}^*\mathbf{Q}_\mathcal{T}$. Next let $\mathbf{V}_\mathcal{T}(t)$ be defined analogously to $\mathbf{P}_\mathcal{T}$, a 3x2 matrix containing the unknown interpolated vertex locations, $v_{\{i,j,k\}}(t)$ at time t. The relationship between these vertex positions and the original positions, $\mathbf{P}_\mathcal{T}$, also uniquely determines a Jacobian, $\mathbf{B}_\mathcal{T}(t)$, which can be written $\mathbf{B}_\mathcal{T}(t) = \mathbf{P}_\mathcal{T}^*\mathbf{V}_\mathcal{T}(t)$.

We obtain a set of desired target Jacobians, $\mathbf{A}_\mathcal{T}(t)$, using the polar decomposition as explained in Section 3.1, and then minimize (1) as in the original method. The notation just presented enables writing the equation more explicitly. As noted in Section 3.2, for consistency we also weight the equations by triangle areas. The final weighted least squares problem becomes:

$$E(t) = \sum_{\mathcal{T}=1}^{M} a_\mathcal{T} \|\mathbf{P}_\mathcal{T}^*\mathbf{V}_\mathcal{T} - \mathbf{A}_\mathcal{T}(t)\|^2, \quad (3)$$

where $a_\mathcal{T}$ is the area of triangle $\mathcal{P}_\mathcal{T}$. Rewriting each expression in terms of a global $N \times 2$ matrix of unknowns, $\mathbb{V}$, we obtain:

$$E(t) = \sum_{\mathcal{T}=1}^{M} a_\mathcal{T} \|\mathbb{P}_\mathcal{T}\mathbb{V} - \mathbf{A}_\mathcal{T}(t)\|^2, \quad (4)$$

where for a triangle index $\mathcal{T}$, with $\texttt{vertices}(\mathcal{T}) = \{i, j, k\}$, $\mathbb{P}_\mathcal{T}$ is a sparse $2 \times N$ matrix with non-zero entries only in columns i, j, k:

$$\mathbb{P}_\mathcal{T} = \begin{bmatrix} .. & \overset{i}{[\mathbf{P}_\mathcal{T}^*]_{11}} & .. & \overset{j}{[\mathbf{P}_\mathcal{T}^*]_{12}} & .. & \overset{k}{[\mathbf{P}_\mathcal{T}^*]_{13}} & .. \\ .. & [\mathbf{P}_\mathcal{T}^*]_{21} & .. & [\mathbf{P}_\mathcal{T}^*]_{22} & .. & [\mathbf{P}_\mathcal{T}^*]_{23} & .. \end{bmatrix}.$$

With this, we can now express the problem in terms of normal equations. We wish to find a minimum residual solution to $\mathbb{P}\mathbb{V} = \mathbb{A}$ with a given weighting. The normal equations that express that are:

$$(\mathbb{P}^T\mathbf{W}\mathbb{P})\mathbb{V} = \mathbb{P}^T\mathbf{W}\mathbb{A}$$

where

$$\begin{aligned} \mathbb{P} &= \begin{bmatrix} \mathbb{P}_1^T & \cdots & \mathbb{P}_M^T \end{bmatrix}^T \\ \mathbb{A} &= \begin{bmatrix} \mathbf{A}_1^T & \cdots & \mathbf{A}_M^T \end{bmatrix}^T \\ \mathbf{W} &= \text{diag}(\begin{bmatrix} a_1 & \cdots & a_M \end{bmatrix}) \end{aligned}$$

In terms of implementation, the core routine consists of assembling the global $N \times N$ matrix $\mathbb{P}^T\mathbf{W}\mathbb{P}$ and $N \times 2$ matrix $\mathbb{P}^T\mathbf{W}\mathbb{A}$. These are precisely the $-\mathbf{H}$ and $\mathbf{G}$ matrices of Alexa et al., respectively. It can be seen that these matrices have the form $\sum_\mathcal{T} a_\mathcal{T}\mathbb{P}_\mathcal{T}^T\mathbb{P}_\mathcal{T}$ and $\sum_\mathcal{T} a_\mathcal{T}\mathbb{P}_\mathcal{T}^T\mathbb{A}_\mathcal{T}$, in other words a sum of very sparse matrix multiplies. Thus implementation consists of just computing small dense products of the form $a_\mathcal{T}(\mathbf{P}_\mathcal{T}^*)^T(\mathbf{P}_\mathcal{T}^*)$ and $a_\mathcal{T}(\mathbf{P}_\mathcal{T}^*)^T\mathbf{A}_\mathcal{T}$ and then scattering the computed elements to their appropriate destinations in the global LHS and RHS matrices, which are sparse. This is identical to the "global assembly" stage of a finite element method.

5 Inexpensive symmetric interpolation

One desirable property of an interpolation technique is *symmetry*. By this it is meant that interpolation $\mathcal{P} \rightarrow \mathcal{Q}$ generates the exact same interpolation as $\mathcal{Q} \rightarrow \mathcal{P}$ under the substitution $t \leftarrow (1 - t)$. [Alexa et al. 2000] presents a modification to the cost equation that makes interpolation symmetric:

$$E(t) = (1 - t)\ \vec{E}\ (t) + t\ \overleftarrow{E}\ (1 - t);$$

however, modifying the cost in this way requires re-solving the linear system for each new t value rather than reusing the fixed, precomputed LU decomposition. By formulating as a least squares problem it is easy to see that one can symmetrize the cost function simply by including both sets of constraint equations in a single matrix. I.e. we wish to find a minimum-residual solution to:

$$\begin{bmatrix} \vec{\mathbb{P}} \\ \overleftarrow{\mathbb{P}} \end{bmatrix} \mathbb{V} = \begin{bmatrix} \vec{\mathbb{A}} \\ \overleftarrow{\mathbb{A}} \end{bmatrix}$$

Package	Size	Factor (μsec)	Solve (μsec)	Factor/Solve
LAPACK[1]	328×328	10506	646.56	16.2
TAUCS[2]	328×328	3249	165.68	19.6
SuperLU[3]	328×328	3005	160.85	18.7
UMFPACK[4]	328×328	3022	133.59	22.6
LAPACK	164×164	2264	65.45	34.6
TAUCS	164×164	1110	72.25	15.4
SuperLU	164×164	638	39.66	16.1
UMFPACK	164×164	1185	38.88	30.5

Table 1: ***Computation Time:*** *Time required to factor and solve two sparse systems of equations using different numerical packages. The 328×328 matrix has 3228 nonzero elements, 164×164 has 892. LAPACK is a dense solver, the others are direct, sparse solvers. (Measurements made on a Pentium IV XEON 3.60Ghz processor.) 1. http://www.netlib.org/lapack/ 2. http://www.tau.ac.il/~stoledo/taucs/ 3. [Demmel et al. 1999] 4. [Davis 2004]*

in the weighted least-squares sense. This still leads to a system matrix of the exact same size and sparsity, but creates a symmetric interpolation that needs be factored only once for all t. This is very significant as can be seen in Table 1, since factoring requires around 15-30 times as much computation as back-substitution.

The key difference is that [Alexa et al. 2000] interpolate the two costs. The approach just explained is equivalent to simply adding the two costs together, as $E(t) = \overrightarrow{E}(t) + \overleftarrow{E}(1-t)$. See Figure 4 for a comparison of interpolations with and without the symmetric cost.

6 Constraints and control

One limitation of the techniques of [Alexa et al. 2000] and [Xu et al. 2005] is that they offer no means for control. Given start and end configurations, the math inexorably leads to a unique set of vertex trajectories. There is no recourse if these trajectories are not satisfactory to the animator. We present some simple and direct methods to influence the interpolation while still attempting to preserve rigidity. These take the form of hard or soft linear constraints, which are easily expressed using the normal equations formulation.

For hard linear constraints there are two main options: substitution and Lagrange multipliers. Substitution is optimal in that it actually reduces the number of degrees of freedom in the system, however it requires that you express your equations exclusively in terms of these remaining degrees of freedom. For this reason, Lagrange multipliers are generally considered to be easier to work with in terms of modular software design. To enforce some additional linear constraints, $\mathbf{C}\mathbb{V} = \mathbf{D}$, using Lagrange multipliers one forms the augmented system:

$$\begin{bmatrix} \mathbb{P}^T\mathbf{W}\mathbb{P} & \mathbf{C}^T \\ \mathbf{C} & 0 \end{bmatrix} \mathbb{V} = \begin{bmatrix} \mathbb{P}^T\mathbf{W}\mathbb{A} \\ \mathbf{D} \end{bmatrix} \tag{5}$$

Some simple possible constraints are

- $\mathbf{C} = [\ldots 0\ 1\ 0 \ldots]$ enables the constraining of a single vertex location.
- $\mathbf{C} = [1/N\ 1/N \ldots]$ constrains the mean of all the vertices.
- $\mathbf{C} = [\ldots 1 \ldots -1 \ldots]$ to enforce that the *difference* between two vertices have a particular value.

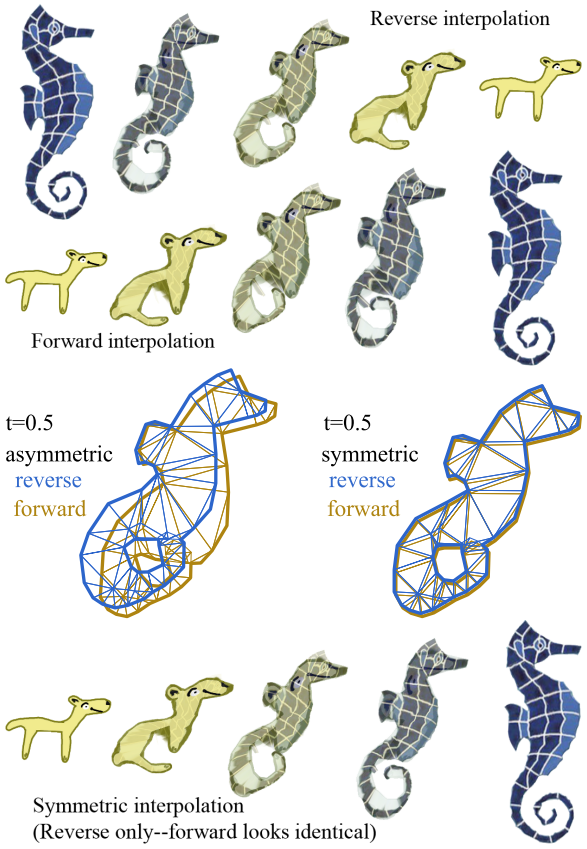

Figure 4: ***Symmetric interpolation:*** *A cartoon animal morphs into a seahorse. With the basic method (top), interpolations are not symmetric in the inputs. A→B generates, to varying degrees, different vertex paths than B→A. We present a simple inexpensive way to make rigid interpolation symmetric (bottom). The difference can be seen clearly by comparing the silhouettes at $t = 0.5$ (middle row).*

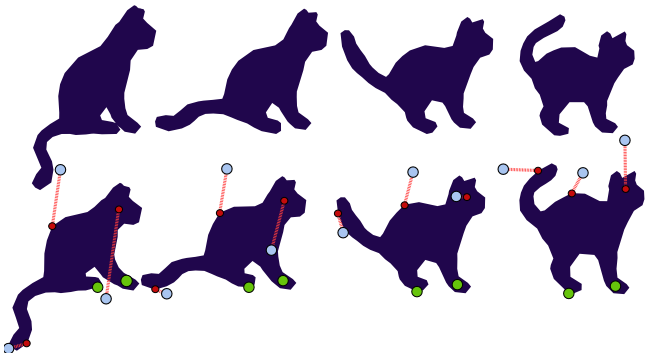

Figure 5: ***Exercising control:*** *Previous rigid interpolation techniques were black-boxes. No means of influencing the resulting interpolation was available. The original sequence (top) was modified using a combination of the hard (green) and soft (blue) constraints we propose in Section 6, resulting in the bottom sequence.*

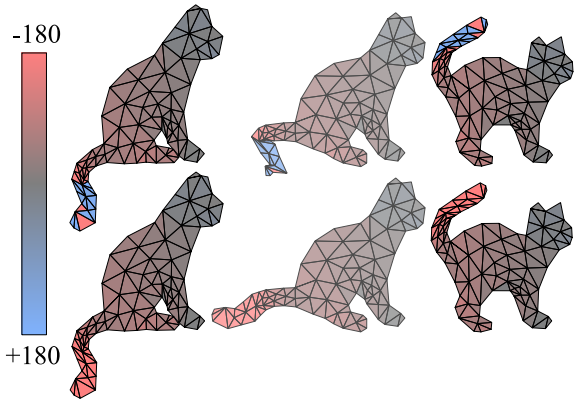

Figure 6: ***Rotation consistency:*** *The coloration indicates clockwise (red) and counter-clockwise (blue) rotation magnitudes computed by per-triangle polar decompositions. Using the original calculation, some neighboring triangles rotate counter to others (top). After applying our scheme, orientations are consistent (bottom).*

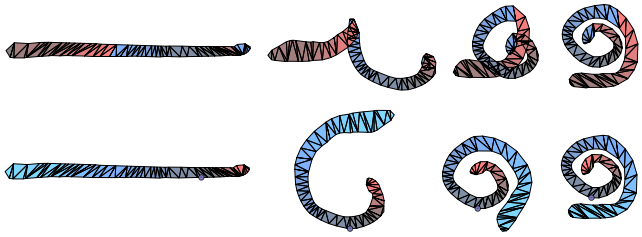

Figure 7: ***Additional large rotation example:*** *A snake coils up. A spread of over 540 degrees is needed between the triangles at the head and tail of the snake. (top) Before applying rotation fix (bottom) after.*

Soft constraints are also easy to incorporate. For these we need a penalty weight, which is easy using the weighted least squares formulation. For this, constraints are expressed by adding additional rows to $\mathbb{P}$. To impose soft constraints $\mathbf{F}\mathbb{V} = \mathbf{G}$:

$$\begin{bmatrix} \mathbb{P}^T & \mathbf{F}^T \end{bmatrix} \mathbf{W} \begin{bmatrix} \mathbb{P} \\ \mathbf{F} \end{bmatrix} \mathbb{V} = \begin{bmatrix} \mathbb{P}^T & \mathbf{F}^T \end{bmatrix} \mathbf{W} \begin{bmatrix} \mathbb{A} \\ \mathbf{G} \end{bmatrix} \quad (6)$$

The strength of the penalty forces can be altered by changing the corresponding weights along the diagonal of $\mathbf{W}$.

These two simple modifications allow us to exercise arbitrary control over individual vertices or collections of vertices while letting the as-rigid-as-possible technique determine the remaining trajectories. Please see Figure 5 and the supplemental video for demonstrations.

7 Making rotation consistent

In a significant number of practical situations, orientation may be interpolated incorrectly by current rigid interpolation approaches such as [Alexa et al. 2000; Xu et al. 2005]. The difficulty stems from the inherent ambiguity of extracting a rotation angle from a rotation matrix. The resulting angle is not unique: $\alpha + 2k\pi$ for $k \in \mathbb{Z}$. The result of this can most clearly be seen in Figures 6 and 7. Looking at the per-triangle rotations, large discontinuities can be seen where the rotation direction changes abruptly because of naïvely choosing the smallest magnitude rotation for each individual triangle.

Some means is needed to make these rotations consistent. Inconsistent rotations can occur anywhere in a shape, either in the interior or on the boundary. To prove this, consider a single triangulated shape and a copy of it rotated $180 - \epsilon$ degrees. Now rotate any triangle of the copy by 2ϵ degrees. This will introduce a rotational discontinuity in the neighborhood of that triangle. In fact, Figure 1 results from a closely related scenario: a copy of the turtle has been rotated by exactly 180 degrees to create the target shape. Due to numerical imprecision, however, triangles end up with rotations of $180 \pm \epsilon$. The situation also arises in less degenerate situations as well (Figures 6, 7). In particular, any interpolation where the rotation required is greater than 180 degrees is problematic, since, absent other information, angle extraction techniques typically return an answer on the $[-180, 180]$ domain.

Figure 8: ***Minimizing rotation:*** *Results like the above can arise from naïve application of the rotation consistency fix. The entire cat rotates around its tail because of an unlucky choice of initial triangle. Compare with corrected minimal rotation sequence in Figs. 5 and 6.*

Our approach is to start from the boundaries and work inward. We wish to find places where the angle extracted from polar decompositions have large discontinuities between adjacent triangles (we use 180 degrees as a threshold), and add or subtract multiples of 360 degrees as needed to get all jumps below 180 degrees. Walking first around the boundary gives us a linear sequence of angles which is quick and easy to process.

Removing discontinuities in this sequence of boundary rotations is in fact a classic problem. It is precisely the problem faced when extracting the phase from a discrete Fourier transform. Most libraries that provide a 1D FFT function also provide some sort of phase `unwrap()` function which does exactly what we need. We put our rotations into an array and just pass it to such a function. The basic algorithm is $O(N)$ and just involves scanning through the values one-by-one looking for discontinuities, and raising or lowering the tail end as needed to eliminate discontinuities greater than 180 degrees.

Once we obtain the new boundary rotations via the FFT phase `unwrap()` procedure, we treat these rotations as ground truth and proceed to propagate these "good" rotations inward in a breadth first manner.

Specifically, we start by pushing all boundary triangles on a stack, tagging them all as *VISITED*. We then iteratively pop them off one by one. For each triangle popped off, we examine its neighbors, and any which are *VISITED* are ignored. For the others we adjust the rotations of any that are inconsistent with the popped triangle, set their *VISITED* flags and push each of them back on the stack. The process continues until the stack is empty. The corrected results in Figures 1, 6 and 7 were generated in this way.

After this is complete, the rotations should be consistent (if a consistent assignment is possible); however, the rotation may not be minimal, as can be seen in Figure 8. Furthermore the presence or lack of excess rotation depends upon the choice of initial triangle. This is true of [Choi and Szymczak 2003] as well. Such dependence is undesirable and should be eliminated. The extra rotation itself may or may not be desired by the user; however, we believe

the minimal rotation is likely to be the one desired in the vast majority of cases, and thus makes the best default.

We use a very simple approach to eliminate the excess rotation. We simply take the area-weighted average rotation over all the triangles and add or subtract multiples of 360 until the average is as close to zero as possible. For the area of triangle $\{i, j, k\}$, we use the average of triangle areas $\{p_i, p_j, p_k\}$ and $\{q_i, q_j, q_k\}$.

8 Discussion and future work

We have presented both a new mathematical formulation and analysis of rigidity preserving interpolation. We pointed out the equivalence of Alexa et al.'s least squares solution and Xu et al.'s Poisson formulation. We additionally identified several areas in which the methods could be improved and offered solutions providing: tessellation-independence (adapted from Xu et al.), inexpensive symmetric cost, direct vertex control of results, and finally we fixed a significant issue with the handling of large rotations.

There is still room for improvement, however. In particular, with our constraint technique, the underlying ideal rotations $\mathbf{A}_{\mathcal{T}}$ do not "know" about the constraints we have imposed, and thus unnatural behavior will result if the constraints are not somewhat consistent with the underlying rotations. Thus there is a limit to the amount of control offered by this technique.

In the future we are interested in pursuing a least-squares formulation of 3D mesh interpolation using the same connection between least-squares and the Poisson equation exploited here for 2D. With the recent popularity of Poisson-based techniques, it also seems likely that many other recent works could take advantage of the simpler, but mathematically equivalent, weighted least squares formulation. Finally, we are currently pursuing ideas for how to provide more flexible rigidity-preserving animation controls.

Acknowledgments: This work was supported in part by the Japan Science and Technology Agency, CREST project.

A Linear least squares and normal equations

Since it is the basis of our reformulation, we present here a brief review some fundamental aspects of linear least squares. More information can be found in [Pighin and Lewis 2007].

Given a set of over- or under-constrained linear equations in unknowns, u, we may express them in matrix form as $Au = b$, where A is non-square. We can find a solution that minimizes $\|Au - b\|^2 = \sum_i \|[A]_i u - [b]_i\|^2$ via the normal equations: $A^T Au = A^T b$. $(A^T A)$ is square, so can be inverted if non-singular or decomposed to find u. In weighted least squares, we wish to minimize $\sum_i w_i \|[A]_i u - [b]_i\|^2$. This can also be expressed in matrix form as $A^T WAu = A^T Wb$, where W is a diagonal matrix with $[W]_{ii} = w_i$. Note that u and b need not be simple vectors—they may be matrices as well.

References

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *Proc. SIGGRAPH '00*, 157–164.

ALEXA, M. 2002. Recent advances in mesh morphing. *Computer Graphics Forum 21*, 2, 173–197.

BEIER, T., AND NEELY, S. 1992. Feature-based image metamorphosis. In *Proc. SIGGRAPH '92*, ACM Press, New York, NY, USA, 35–42.

CHOI, J., AND SZYMCZAK, A. 2003. On coherent rotation angles for as-rigid-as-possible shape interpolation. In *Proc. 15th Canadian Conf. Comp. Geom.*, 111–114.

DAVIS, T. A. 2004. Algorithm 832: Umfpack v4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw. 30*, 2, 196–199.

DEMMEL, J. W., EISENSTAT, S. C., GILBERT, J. R., LI, X. S., AND LIU, J. W. H. 1999. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications 20*, 3, 720–755.

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

GOTSMAN, C., AND SURAZHSKY, V. 2001. Guaranteed intersection-free polygon morphing. *Computers and Graphics 25*, 1, 67–75.

GREGORY, A., STATE, A., LIN, M., MANOCHA, D., AND LIVINGSTON, M. 1998. Feature-based surface decomposition for correspondence and morphing between polyhedra. In *CA '98: Proceedings of the Computer Animation*, 64.

KANAI, T., SUZUKI, H., AND KIMURA, F. 2000. Metamorphosis of arbitrary triangular meshes. *IEEE Comp. Graph. App. 20*, 2, 62–75.

KAVAN, L., DOBBYN, S., COLLINS, S., ZARA, J., AND O'SULLIVAN, C. 2008. Polypostors: 2d polygonal impostors for 3d crowds. In *2008 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press.

LEE, S.-Y., CHWA, K.-Y., AND SHIN, S. Y. 1995. Image metamorphosis using snakes and free-form deformations. In *Proc. SIGGRAPH '95*, 439–448.

LEE, A. W. F., DOBKIN, D., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution mesh morphing. In *Proc. SIGGRAPH '99*, 343–350.

PIGHIN, F., AND LEWIS, J. P. 2007. Practical least-squares for computer graphics. In *SIGGRAPH '07: courses*, ACM, New York, NY, USA, 1–57.

SCHAEFER, S., MCPHAIL, T., AND WARREN, J. 2006. Image deformation using moving least squares. In *Proc. SIGGRAPH '06*, 533–540.

SHEFFER, A., AND KRAEVOY, V. 2004. Pyramid coordinates for morphing and deformation. In *Proc. 2nd International Symposium on 3D Data Processing, Visualization, and Transmission*.

SUMNER, R. W., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. 2005. Mesh-based inverse kinematics. *Proc. SIGGRAPH '05*, 488–495.

SURAZHSKY, V., AND GOTSMAN, C. 2001. Controllable morphing of compatible planar triangulations. *ACM Transactions on Graphics 20*, 4, 203–231.

TONG, Y., LOMBEYDA, S., HIRANI, A. N., AND DESBRUN, M. 2003. Discrete multiscale vector field decomposition. *Proc. SIGGRAPH 03 22*, 3, 445–452.

WOLBERG, G. 1998. Image morphing: A survey. *The Visual Computer 14*, 8, 360–372.

XU, D., ZHANG, H., WANG, Q., AND BAO, H. 2005. Poisson shape interpolation. In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, 267–274.